AFRL-RI-RS-TR-2017-246

INFERENCE FOR CONTINUOUS-TIME PROBABILISTIC PROGRAMMING

UNIVERSITY OF CALIFORNIA AT RIVERSIDE

DECEMBER 2017

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2017-246 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /	**/ S /**
ROGER J. DZIEGIEL JR.	JON S. JONES
Work Unit Manager	Technical Advisor, Information Intelligence Systems and Analysis Division
	Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* DECEMBER 2017	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* OCT 2013 – AUG 2017

4. TITLE AND SUBTITLE INFERENCE FOR CONTINUOUS-TIME PROBABILISTIC PROGRAMMING	**5a. CONTRACT NUMBER** N/A
	5b. GRANT NUMBER FA8750-14-2-0010
	5c. PROGRAM ELEMENT NUMBER 61101E
6. AUTHOR(S) Christian R. Shelton	**5d. PROJECT NUMBER** PPML
	5e. TASK NUMBER 3U
	5f. WORK UNIT NUMBER CR
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of California at Riverside, 200 University Office Bldg., Riverside CA 92521	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RIED, 525 Brooks Road, Rome NY 13441-4505; DARPA, 675 North Randolph St, Arlington, VA 22203-2114	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RI
	11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2017-246

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Machine learning – the ability of computers to understand data, manage results and infer insights from uncertain information – is the force behind many recent revolutions in computing. Email spam filters, smartphone personal assistants and self-driving vehicles are all based on research advances in machine learning. Unfortunately, even as the demand for these capabilities is accelerating, every new application requires a Herculean effort. Teams of hard-to-find experts must build expensive, custom tools that are often painfully slow and can perform unpredictably against large, complex data sets. This project developed new algorithms for statistical inference in continuous-time probabilistic models. This report first reviews background on continuous-time models and then covers each of the research projects and their deliverables. The full details are covered in the technical papers, included as appendices.

15. SUBJECT TERMS

Inference Algorithms, Machine Learning, Probabilistic Programming

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** ROGER J. DZIEGIEL, JR.
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	154	**19b. TELEPHONE NUMBER (*Include area code*)** N/A

Table of Contents

Continuous-Time Probabilistic Models

1 Summary

This project developed new algorithms for statistical inference in continuous-time probabilistic models. This report first reviews background on continuous-time models and then covers each of the research projects and their deliverables. The full details are covered in the technical papers, included as appendices.

2 Introduction

A continuous-time probabilistic model describes a distribution over a set of events in time. That is, a sample from this distribution consists of a set of events (whose number is random), where each event is associated with a real-valued time and (possibly) additional information pertaining to the event.

For instance, events might be posts in a social network [1, 2, 3, 4], armed conflicts [5, 6, 7], earthquakes [8], or robot motor or sensor malfunctions [9]. The stochastic model describes the distribution over the number of events, their times, and additional information (such as location, type, user).

While a standard probability model describes the probability of an event, possibly conditioned on other factors, as the time of event can be continuous, continuous-time models describe the probabilistic rate of an event. A general form is as an intensity function $\lambda(t, h_t)$ where t is the time and h_t is the history of events prior to time t. $\lambda(t, h_t)$ is the instantaneous rate of an event at time t: the (instantaneous) probability per unit time of an event. If λ is a constant, then the model is known as a Poisson process: The interarrival time between events is exponentially distributed, and the events in non-overlapping windows of time are independent. While simple, this model does not have predictive power.

A wide variety of continuous-time process models have been invented including Poisson-networks [10], piecewise-constant conditional intensity models [11], continuous-time Bayesian networks [12], and Hawkes processes [13]. We developed new inference methods for each of the last three, and will, therefore, describe them in a bit more detail below.

2.1 Markovian Processes

Continuous-time Bayesian networks (CTBNs) are Markovian models. That is, they assume that the information in the past needed to form the distribution of future events can be summarized by a "state." For a CTBN, this state is an assignment of values to variables. A CTBN is an example of a continuous-time Markov process (or chain). The distinction is that a CTBN provides a compact representation of the process when the number of variables is large (as the total number of states in the system grows exponentially with the number of variables).

The events in a CTBN correspond to changes in the state variables and the function λ is a function of the state variables only (or the most recent events for each such variable). The intensity function is piecewise-constant, with changes only at events. This leads to exponential inter-arrival times between events. Hiding (or masking) certain variables leads to more interesting non-Markovian behavior over the other variables (if the hidden variables are marginalized out).

There have been a wide variety of inference methods developed for CTBNs including variational approaches [14, 15], sampling methods [16, 17], and others [18, 19, 20]. The sampling methods converge (in the limit of infinite samples), but are random algorithms. The other methods are non-random, but do not converge.

2.2 Piecewise-constant Conditional Intensity Models

Piecewise-constant conditional intensity models (PCIMs) are models in which the function $\lambda(t, h_t)$ can be described as a decision tree. This causes λ to be a piecewise-constant function (of time) for any fixed history. The decision nodes in the tree are functions of the history (such as, "were the last two events both in the past 1 hour?"). This allows for both self-suppression and self-excitation. If there are multiple types of events (that is, events with different "additional information"), PCIMs model each with their own tree, but the decisions in one tree can depend upon the history of events from another tree.

PCIMs have efficient learning algorithms, for building the decision trees from data, when all events have been observed. They also have efficient forward sampling algorithms: methods for extending a sample forward in time. The missing piece was being able to sample conditioned on future data, to answer questions about what might have happened to cause observed events.

2.3 Hawkes Processes

A Hawkes process (HP) [13] (rediscovered as a cascade of Poisson processes [1]) describes the intensity function, λ as a sum of a base constant rate, plus a contribution for each event in the past. The contribution of an event Δt time units ago is captured by a kernel function, $\phi(\Delta t)$ which is the key ingredient in the HP model.

An HP can describe self-excitatory processes (but not self-inhibitory ones). The rate of an event is also unbounded, leading it to be a good model for many types of viral models. Prior to this work, parameter (kernel) estimation methods were known, but all methods assumed that all events were observed. No one had considered the calculation of posteriors over unobserved historic events.

3 Methods, Assuptions, and Procedures

Our proposal organized the work into two categories: Markovian processes and non-Markovian processes. For the first two years of the project, we focused mainly on the Markovian processes, whereas we flipped the focus in the last two years.

3.1 Markovian Processes

A Markovian continuous-time process can be described by a rate matrix, Q, where element q_{ij} is the rate of transitioning from state i to state j. For systems in which a state is an assignment of values to variables, the size of this matrix grows exponentially with the number of variables. Continuous-time Bayesian networks (CTBNs) solve this problem by introducing a graph over the variables to describe which variables immediately influence which others' transition rates. The result, much like that for standard Bayesian networks, is that the descriptive space for the model grows exponentially with the fan-in of the graph, but only linearly with the number of variables.

The key computational operation for reasoning about a Markovian continuous-time system is the matrix exponential. In particular, to compute the marginal distribution of the state at one time given its value at a previous time, we need to exponentiate the matrix Q times the time duration. For a CTBN (or any other compact model), this cannot be done efficiently and exactly, unless P=NP. As mentioned above, many approximate methods have been developed.

3.2 Piecewise-constant Conditional Intensity Models

Piecewise-constant conditional intensity models (PCIMs) represent the intensity function as a decision tree where the decisions at inner nodes are based on the raw clock time and also the history of events. As this tree grows, it can approximate a class of "universal" models [25] and has been extended to a multiplicative forest model [26, 27].

We have developed new inference methods that allow for hidden, unseen, event types. These events might represent actions taken by unobserved actors (in a social network, for instance), abstract change points in the dynamics, or high-level human-described labels. Previously these types of events have been ignored by non-Markovian models, figuring that the non-Markovian aspect could account for the patterns in the observed data without the need for unobserved explanations. We demonstrated the advantage of such unobserved event types in PCIMs and Hawkes processes.

3.3 Hawkes Processes

A Hawkes process is also a non-Markovian process. However, the intensity function is a sum, over all previous events. This means that its value is unbounded. The method we developed for PCIMs depended on a finite upper limit to the intensity function. Yet, Hawkes processes have become very popular in modeling earthquake modeling [8], finance [7, 32], social networks [1, 2, 3, 4], influence maximization [33], armed conflicts [5, 6, 7], and topic modeling [34, 35, 36]. Therefore, we wished to develop the first general inference method for Hawkes processes.

While other applications placed priors on the parameters of the Hawkes process or other parts of the model and used Gibbs sampling (or similar) over these parameters [7, 37], no work to date has considered unobserved events to the extent that we do in this paper. [38] used mixtures of Hawkes processes and developed a variational inference method. However, all events were observed; only their assignments to mixture components were not. [39] considered the situation in which all events prior to a specific time are unobserved (left censoring), which is a very specific case of the missingness patterns we allow. Finally, if there are no observed events of any labels and the process has an exponential kernel, a closed-form solution exists [33] However, no general closed-form solution is known for the evidence patterns we consider here.

4 Results and Discussion

Our methods above yielded the following new algorithms and results.

4.1 Markovian Processes

Our work developed the first anytime, convergent, non-random method. Previous sampling methods were anytime and convergent, but they would give different results each run. Previous non-random methods did not converge. Our method is based on a decomposition of the matrix exponential into a series of nested integrals, known as a time-ordered product. It was originally developed in the context of quantum electrodynamics [21, 22].

While the integrals must be approximated, their approximations can be reused in evaluating the next, nested, integral. The result is a computation tree which we traverse in a greedy fashion to build up an answer. The full details and comparisons to previous methods can be found in [23, 24].

In the proposal, an additional method, using ideas from Gibbs sampling, was to be developed. We worked on this for approximately one year without success. In the meanwhile, we continued on the work described below and found new avenues for research, not part of the original proposal, and instead developed those, resulting in the Hawkes process inference method detailed below.

4.2 Piecewise-Constant Conditional Intensity Models

Our inference algorithm permits the reasoning about such missing events and the estimation (learning) of a model from data that is missing events. The algorithm is a generalization of a previous Markovian algorithm [28], extending it to the PCIM case.

We used this method to beat state-of-the-art methods in video activity recognition [29, 30, 31]. The PCIM model can capture temporal dependencies between and among event types and visual features. The result is more adaptive and has better accuracy than existing methods.

4.3 Hawkes Processes

We used a combination of the auxiliary variables (a mathematical technique from our PCIM approach) with the generational view of Hawkes processes to develop a Markov chain Monte Carlo sampler for Hawkes processes under the same unobserved event assumptions as the PCIM sampler. We demonstrated its utility in analyzing city-level crime data to find sociologically known patterns that were not revealed without the hidden event types [40].

This method was not in the original proposal, but was pursued (with program manager's agreement) after initial results on PCIM inference yielded good results and our research into non-Markovian processes suggested a Hawkes process method would be successful. Further, since the proposal, Hawkes processes had become more popular and better suited to a number of emerging applications, including social media tracking.

5 Conclusions

The code for these projects, so that they can be reproduced and built upon, is available through github on the following projects:

- cshelton/comtope
- cshelton/pcim-plain
- cshelton/hawkesinf
- pierce1987/pcim-dev

The technical details of these methods and their results on the applications are given the academic papers attached to this report.

This project almost uniformly achieved the proposal's goals. In Markov inference, we did not successfully develop the second set of goals. However, we were able to achieve an unproposed inference method for Hawkes processes, a model that had become popular since the time of the proposal. All other proposed research was carried out successfully.

6 References

[1] A. Simma and M. Jordan. Modeling events with cascades of Poisson processes. In *UAI*, 2010.

[2] Ke Zhou, Honghuan Zha, and Le Song. Learning triggering kernels for multi-dimensional Hawkes processes. In *ICML*, volume 28, 2013.

[3] Patrick O. Perry and Patrick J. Wolfe. Point process modelling for directed interaction networks. *Journal of the Royal Statistical Society: Series B*, 75(5):821–849, 2013.

[4] Christopher DuBois, Carter T. Butts, and Padhraic Smyth. Stochastic blockmodeling of relational event dynamics. In *AI-STATS*, 2013.

[5] Charles Blundell, Katherine A. Heller, and Jeffrey M. Beck. Modelling reciprocating relationships with Hawkes processes. In *NIPS*, 2012.

[6] George Mohler. Modeling and estimation of multi-source clustering in crime and security data. *The Annals of Applied Statistics*, 7(3):1525–1539, 2013.

[7] Scott W. Linderman and Ryan P. Adams. Discovering latent network structure in point process data. In *ICML*, 2014.

[8] David Marsan and Olivier Lengliné. Extending earthquakes' reach through cascading. *Science*, 319:1076–1079, February 2008.

[9] Brenda Ng, Avi Pfeffer, and Richard Dearden. Continuous time particle filtering. In *AAAI*, pages 1360–1365, 2005.

[10] Shyamsundar Rajaram, Thore Graepel, and Ralf Herbrich. Poisson networks: A model for structured point processes. In *Proceedings of the AI STATS 2005 Workshop*, 2005.

[11] A. Gunawardana, C. Meek, and P. Xu. A model for temporal dependencies in event streams. In *NIPS*, 2011.

[12] Uri Nodelman, Christian R. Shelton, and Daphne Koller. Continuous time Bayesian networks. In *Proceedings of the Eighteenth International Conference on Uncertainty in Artificial Intelligence*, pages 378–387, 2002.

[13] Alan G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[14] Ido Cohn, Tal El-Hay, Raz Kupferman, and Nir Friedman. Mean field variational approximation for continuous-time Bayesian networks. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*, 2009.

[15] Tal El-Hay, Ido Cohn, Nir Friedman, and Raz Kupferman. Continuous-time belief propagation. In *Proceedings of the 27th International Conference on Machine Learning*, pages 343–350, Haifa, Israel, June 2010.

[16] Yu Fan and Christian R. Shelton. Sampling for approximate inference in continuous time Bayesian networks. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics*, 2008.

[17] Yu Fan, Jing Xu, and Christian R. Shelton. Importance sampling for continuous time Bayesian networks. *Journal of Machine Learning Research*, 11(Aug):2115–2140, 2010.

[18] Uri Nodelman, Christian R. Shelton, and Daphne Koller. Expectation maximization and complex duration distributions for continuous time Bayesian networks. In *Proceedings of the Twenty-First International Conference on Uncertainty in Artificial Intelligence*, pages 421–430, 2005.

[19] Suchi Saria, Uri Nodelman, and Daphne Koller. Reasoning at the right time granularity. In *Proceedings of the Twenty-third Conference on Uncertainty in AI*, pages 421–430, 2007.

[20] E. Busra Celikkaya, Christian R. Shelton, and William Lam. Factored filtering of continuous-time systems. In *Proceedings of the Twenty-Seventh International Conference on Uncertainty in Artificial Intelligence*, 2011.

[21] F. J. Dyson. The radiation theories of Tomonaga, Schwinger, and Feynman. *Physical Review*, 75(3):486–502, 1949.

[22] Richard P. Feynman. An operator calculus having applications in quantum electrodynamics. *Physical Review*, 84(1):108–128, 1951.

[23] E. Busra Celikkaya and Christian R. Shelton. Deterministic anytime inference for stochastic continuous-time Markov processes. In *Proceedings of the Thirty-First International Conference on Machine Learning*, 2014.

[24] Emine Busra Celikkaya. *Multivariate Continuous-Time Models: Approximate Inference Algorithms and Medical Informatics Applications.* PhD thesis, University of California at Riverside, March 2016.

[25] Asela Gunawardana and Christopher Meek. Universal models of multivariate temporal point processes. In *AISTATS*, 2016.

[26] Jeremy C. Weiss and David Page. Forest-based point processes for event prediction from electronic health records. In *Proceedings of the European Conference in Machine Learning and Principals and Practice of Knowledge and Discovery in Databases (ECML-PKDD)*, 2013.

[27] Jeremy C. Weiss. Piecewise-constant parametric approximations for survival learning. In *Proceedings of Machine Learning for Healthcare*, 2017.

[28] Vinayak Rao and Yee Whye Teh. Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In *UAI*, 2011.

[29] Zhen Qin. *Modeling Social and Temporal Context for Video Analysis.* PhD thesis, University of California at Riverside, June 2015.

[30] Zhen Qin and Christian R. Shelton. Social grouping for multi-target tracking and head pose estimation in video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2082–2095, October 2016.

[31] Zhen Qin and Christian R. Shelton. Event detection in continuous video: An inference in point process approach. *IEEE Transactions on Image Processing*, 26(12):5680–5691, December 2017.

[32] Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(1), 2015.

[33] Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyan Zha. Scalable influence estimation in continuous-time diffusion networks. In *NIPS*, 2013.

[34] Xinran He, Theodoros Rekatsinas, James Foulds, Lise Getoor, and Yan Liu. HawkesTopic: A joint model for network inference and topic modeling from text-based cascades. In *ICML*, 2015.

[35] Fangjian Guo, Charles Blundell, Hanna Wallach, and Katherine Heller. The Bayesian echo chamber: Modeling social influence via linguistic accommodation. In *AI-STATS*, 2015.

[36] Nan Du, Mehrdad Farajtabar, Amr Ahmed, Alexander J. Smola, and Le Song. Dirichlet-Hawkes processes with applications to clustering continuous-time document streams. In *KDD*, 2015.

[37] Jakob Gulddahi Rasmussen. Bayesian inference for Hawkes processes. *Methodology and Computing in Applied Probability*, 15(3):623–642, September 2013.

[38] Shuang-Hong Yang and Hongyuan Zha. Mixture of mutually exciting processes for viral diffusion. In *ICML*, 2013.

[39] Hongteng Xu, Dixin Luo, and Hongyuan Zha. Learning Hawkes processes from short doubly-censored event sequences. In *ICML*, 2017.

[40] Christian R. Shelton, Zhen Qin, and Chandini Shetty. Hawkes process inference with missing data. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. to appear.

Appendix A

Tutorial on Structured Continuous-Time Markov Processes

Appeared as

Christian R. Shelton and Gianfranco Ciardo. Tutorial on Continuous-Time Markov Processes. *Journal of Artificial Intelligence Research*, 51, 725–778, 2014.

Journal of Artificial Intelligence Research 51 (2014) 725–778 Submitted 04/14; published 12/14

Tutorial on Structured Continuous-Time Markov Processes

Christian R. Shelton CSHELTON@CS.UCR.EDU
University of California, Riverside

Gianfranco Ciardo CIARDO@IASTATE.EDU
Iowa State University

Abstract

A continuous-time Markov process (CTMP) is a collection of variables indexed by a continuous quantity, time. It obeys the Markov property that the distribution over a future variable is independent of past variables given the state at the present time. We introduce continuous-time Markov process representations and algorithms for filtering, smoothing, expected sufficient statistics calculations, and model estimation, assuming no prior knowledge of continuous-time processes but some basic knowledge of probability and statistics. We begin by describing "flat" or unstructured Markov processes and then move to structured Markov processes (those arising from state spaces consisting of assignments to variables) including Kronecker, decision-diagram, and continuous-time Bayesian network representations. We provide the first connection between decision-diagrams and continuous-time Bayesian networks.

1. Tutorial Goals

This tutorial is intended for readers interested in learning about continuous-time Markov processes, and in particular compact or structured representations of them. It is assumed that the reader is familiar with general probability and statistics and has some knowledge of discrete-time Markov chains and perhaps hidden Markov model algorithms.

While this tutorial deals only with Markovian systems, we do not require that all variables be observed. Thus, hidden variables can be used to model long-range interactions among observations. In these models, at any given instant the assignment to *all* state variables is sufficient to describe the future evolution of the system. The variables themselves real-valued (continuous) times. We consider evidence or observations that can be regularly spaced, irregularly spaced, or continuous over intervals. These evidence patterns can change by model variable and time.

We deal exclusively with discrete-state continuous-time systems. Real-valued variables are important in many situations, but to keep the scope manageable, we will not treat them here. We refer to the work of Särkkä (2006) for a machine-learning-oriented treatment of filtering and smoothing in such models. The literature on parameter estimation is more scattered. We will further constrain our discussion to systems with finite states, although many of the concepts can be extended to countably infinite state systems.

We will be concerned with two main problems: inference and learning (parameter estimation). These were chosen as those most familiar to and applicable for researchers in artificial intelligence. At points we will also discuss the computation of steady-state properties, especially for model for which most research concentrates on this computation.

The first section (Section 2) covers the basics of flat (unstructured state-space) continuous-time Markov processes. The remaining sections discuss compact representations. This tutorial's goal is to make the mathematical foundations clear and lay out the current research landscape so that more detailed papers can be read more easily.

1.1 Related Models

There are many non-Markov continuous time models. Gaussian processes (Williams, 1998) are the best-known and model continuous-valued processes. For discrete-valued processes, most models build upon Poisson processes, or more general marked processes. As a Poisson process is memoryless, to make an interesting model, researchers usually generalize to allow the rate of an event to be a function of the process's history.

Poisson networks (Rajaram, Graepel, & Herbrich, 2005) constrain this function to depend only on the counts of the number of events (possibly of different event types) during a finite time window. The cascade of Poisson process model (Rajaram et al., 2005) defines the rate function to be the sum of a kernel applied to each historic event. The kernel has parameters for the effect of time passing, overall event rate, and chance that one type of event follows another. Piecewise-constant intensity models (PCIMs) (Gunawardana, Meek, & Xu, 2012; Parikh, Gunamwardana, & Meek, 2012) define the intensity function as a decision tree, with internal nodes' tests drawn from a set of pre-specified binary tests of the history. Forest-based point processes (Weiss & Page, 2013) extend this by allowing the intensity function to be the product of a set of functions, each a PCIM-like tree. Didelez (2008) presents a generalization of the continuous-time Bayesian networks (see Section 5) to inhomogeneous point processes, but without specific parameterizations or algorithms.

1.2 Why Continuous Time

Contemporary computers are discrete-time computation engines (or at least present a model of one). Therefore, why would we consider a continuous-time model? The quickest answer is by analogy: We build models of non-temporal systems employing real-valued variables. The tools of linear algebra, calculus, and the like allow us to derive and analyze these algorithms and methods. Yet, in the end they will be implemented on discrete-valued computers with finite memory and precision. However, we find the abstraction of continuous-valued variables useful and only make approximations during the final implementation when employing fixed- or floating-point precision arithmetic.

Similarly, it is productive to treat time as a continuous quantity. It allows us to more naturally discuss and reason about systems in which

1. Events, measurements, or durations are irregularly spaced,
2. Rates vary by orders of magnitude, or
3. Durations of continuous measurement need to be expressed explicitly.

All of these happen in asynchronous systems. Most dynamic systems of interest are asynchronous: events or measurements (or both) do not occur based on some global clock. Social networks, phylogenetic trees, and computer system logs are just some examples.

Note that while the underlying system model is continuous-time, observations and measurements of that model need not be continuous. We directly treat discrete-time observations, both at regular and irregular intervals.

1.3 Why Not Discrete Time

Clearly for any given continuous-time system specification, some discretization of time values could be made without introducing too much approximation error. Such a conversion of time from real-valued to integral makes it mathematically more difficult to be flexible about how to treat time algorithmically. This makes the development of computationally efficient algorithms more difficult. For instance, in a discrete-time model, it is natural to have computations proceed one time "step" at a time. However, for uneventful times, this can be computationally overly burdensome. With a continuous-time model, because there is no natural time step, it is simpler to think about methods that can "jump" over such uneventful time periods. Additionally, there are a few oddities about Markov chains built by discretizing continuous time. Finally, the full system specification may not be known when the discretization must be selected (for instance, if parameters must be estimated).

1.3.1 Time Discretization and Markovian-ness

Consider the two-state Markov chain X described by the stochastic matrix[1]

$$\boldsymbol{T_1} = \begin{bmatrix} 0.75 & 0.25 \\ 0.5 & 0.5 \end{bmatrix} . \tag{1}$$

The elements of $\boldsymbol{T_1}$ are the probabilities $p(X_t \mid X_{t-1})$ for each value of X_t and X_{t-1}. Over one time unit, the probability of moving from state 1 to state 2 is 0.25, for example.

If $\boldsymbol{T_1}$ describes a continuous-time system, sampled at a period of 1 time unit, there should be a matrix $\boldsymbol{T_{1/2}}$ describing the same system, sampled at a period of $\frac{1}{2}$ time unit (or twice the sampling *rate*). Indeed there is:

$$\boldsymbol{T_{1/2}} = \begin{bmatrix} 0.83 & 0.17 \\ 0.33 & 0.67 \end{bmatrix} . \tag{2}$$

This can be verified:

$$\begin{aligned} P(X_t = j \mid X_{t-1} = i) &= \sum_k P(X_{t-1/2} = k \mid X_{t-1} = i) P(X_t = j \mid X_{t-1/2} = k) \\ \boldsymbol{T_1}(i,j) &= \sum_k \boldsymbol{T_{1/2}}(i,k) \boldsymbol{T_{1/2}}(k,j) \\ \boldsymbol{T_1} &= \boldsymbol{T_{1/2}} \boldsymbol{T_{1/2}} \ . \end{aligned}$$

That is, $\boldsymbol{T_{1/2}}$ is the matrix square root of $\boldsymbol{T_1}$.

Now take a different two-state Markov chain transition matrix

$$\boldsymbol{S_1} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix} \tag{3}$$

and construct the corresponding transition matrix at half the sampling period, $\boldsymbol{S_{1/2}}$:

$$\boldsymbol{S_{1/2}} = \begin{bmatrix} 0.5 + .447i & 0.5 - .447i \\ 0.5 + .447i & 0.5 - .447i \end{bmatrix} . \tag{4}$$

1. We will use *row*-stochastic matrices exclusively in this tutorial. While column-stochastic matrices are often used for discrete time, row-stochastic matrices are more common in continuous time.

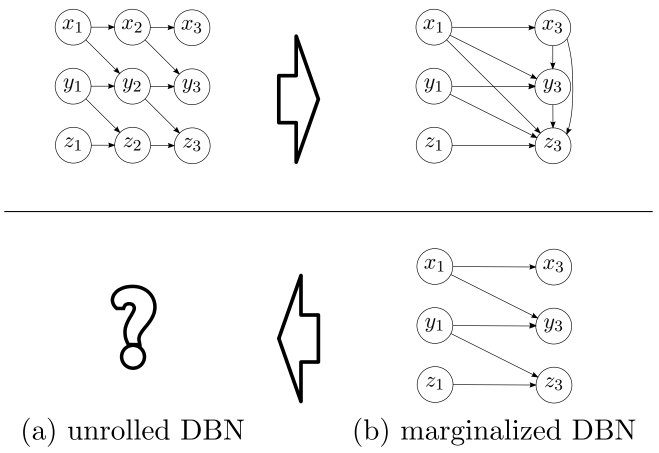


Figure 1: Example of (a) a DBN unrolled, and (b) the same DBN marginalized to twice the sampling periodicity

There is no real-valued stochastic matrix describing the same processes as $\boldsymbol{S_1}$, but at half the sampling periodicity. Put differently, there is no two-state continuous-time Markov system that when sampled at a rate of 1 time unit produces the Markov chain with transition matrix $\boldsymbol{S_1}$.

The problem in generating $\boldsymbol{S_{1/2}}$ arises because $\boldsymbol{S_1}$ has a negative eigenvalue (by contrast, all eigenvalues of $\boldsymbol{T_1}$ are positive). In general, only stochastic matrices with all positive eigenvalues correspond to a continuous-time Markov process sampled at a given periodicity. This can be viewed in two ways. First, it means that the set of continuous-time Markov processes is smaller than the set of discrete-time Markov processes. Second, it means that there are processes that are Markovian only when sampled at a particular periodicity and the only way to extend them to time points outside that periodicity would be to construct a non-Markovian (and non-stationary) process.

If the periodicity of a discrete-time Markov chain is inherent to the process, then this result is not of concern. However, many systems do not have a natural sampling rate. The rate is chosen for computational or measurement convenience. In this case, we must be careful about how we employ our Markovian assumption. Or, we should directly model the underlying system in continuous time.

1.3.2 Independencies and Markovian-ness

A similar problem arises for independencies. We describe the problem here in terms of dynamic Bayesian networks (DBNs) (Dean & Kanazawa, 1989). If unfamiliar with DBNs, the reader may skip to the next section.

Consider the DBN in Figure 1(top,a), which has been unrolled one time step. We can marginalize out the middle time slice and the result is the DBN in Figure 1(top,b): the same model, but over twice the sampling periodicity. However, perhaps we wish to go the opposite direction (to half the sampling periodicity). Figure 1(bottom,b) shows a DBN over

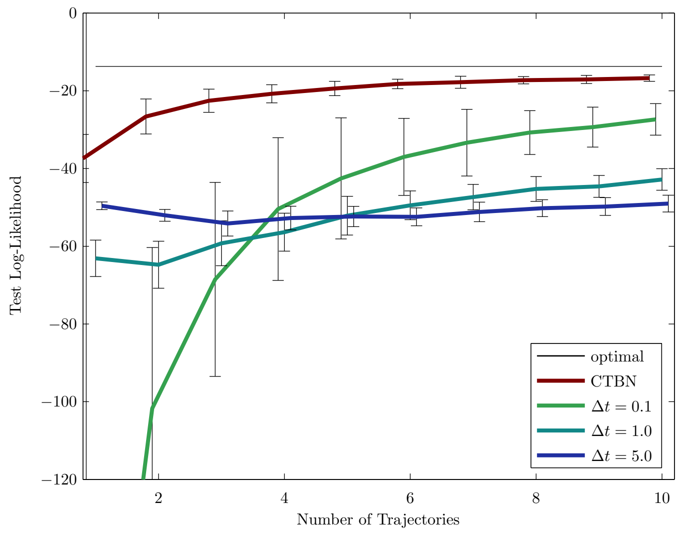


Figure 2: Comparison of learning DBNs with different time-slice widths

two time units. There is no DBN graph structure over one time unit that would marginalize to this graph structure. There may be a DBN, but the independencies expressed by the graph structure over two time units are not expressible in the *graph structure* at half the sampling periodicity. Such independencies would be buried in the parameters of the DBN (if those parameters are possible, given the previous discussion). This means that the independencies expressed by a DBN's graph are a function of both the underlying process and the sampling rate.

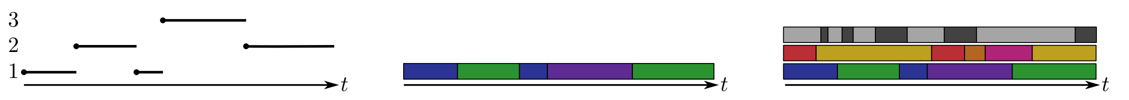
1.3.3 Structure Learning

Selection of a sampling rate is not just a theoretical problem. Nodelman, Shelton, and Koller (2003) demonstrate the problem for parameter estimation. In particular, they considered data drawn from a continuous-time Markovian process of eight variables (mostly binary). The resulting trajectories were discretized in time according to a parameter Δt and DBNs (including structure) were learned for each setting. Figure 2 shows the test log-likelihood accuracy as a function of the number of training trajectories and Δt. It also shows the result of not discretizing time (the CTBN line, a model explained in Section 5).

While it is not too surprising that the CTBN model does the best (as the data were generated from this model), it is instructive that the best Δt depends on the number of observed trajectories. This means that, if the sampling periodicity is a model parameter, its choice cannot be made independently of the amount of data used to estimate the DBN.

2. Continuous-Time Markov Processes

A continuous-time Markov process (CTMP) is a distribution over trajectories. A trajectory (or sample) of a CTMP is a right-continuous piece-wise constant function of a real-valued variable, time. Figure 3 illustrates example trajectories. If the states have a natural order-



(a) ordered states (b) unordered states (c) multiple state variables

Figure 3: Example continuous-time Markov process samples (trajectories)

ing, Figure 3(a) might be a natural depiction. If the states are not ordered, Figure 3(b) depicts the same sample for a three-state system. In later sections we will be considering large factored state spaces in which a state is an assignment to multiple variables. Figure 3(c) depicts such a trajectory.

A finite CTMP defines a set of random variables, each with the same finite sample space (the state space), indexed by a real-value usually denoted as t for time. Let X be such a process. The Markovian property states that

$$X(t_1) \perp X(t_3) \mid X(t_2), \forall\, t_1 < t_2 < t_3 \ . \tag{5}$$

Throughout this tutorial, we will describe distributions over both continuous and discrete random variables. We will use lowercase letters for the densities of continuous random variables and uppercase letters for the probabilities of discrete random variables.

2.1 Parameterization

We will parameterize a CTMP X by a starting distribution at time $t = 0$, $P(X(0))$ (and restrict $t \geq 0$) and an intensity matrix $\boldsymbol{Q_X}$ (or just $\boldsymbol{Q}$ if the context is clear). The starting distribution is just as in a discrete-time Markov chain, and we will largely ignore it. The intensity matrix is analogous to the transition matrix of a discrete-time process.

2.1.1 Comparison to Discrete-Time

Consider the following (roughly equivalent) discrete-time transition matrix $\boldsymbol{M}$ and continuous-time intensity matrix $\boldsymbol{Q}$:

$$\boldsymbol{M} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.1 & 0.7 \end{bmatrix} \qquad \boldsymbol{Q} = \begin{bmatrix} -0.8 & 0.32 & 0.48 \\ 0.12 & -0.24 & 0.12 \\ 0.27 & 0.13 & -0.4 \end{bmatrix} .$$

We can interpret a row of $\boldsymbol{M}$ in two ways. The first row could be viewed as stating that if the process is in state 1, at the next time step there is a 0.5 chance that it will be in state 1, a 0.2 chance that it will be in state 2, and a 0.3 chance that it will be in state 3. Alternatively, it could be viewed as stating that if the process is in state 1, it will remain there for a number of steps geometrically distributed: Pr(stay for n steps) $= 0.5^n$. And, when it leaves it will transition to state 2 with probability $0.2/0.5 = 0.4$ and to state 3 with probability $0.3/0.5 = 0.6$.

The intensity matrix $\boldsymbol{Q}$ has two similar interpretations. The first row states that if the process is in state 1, after a short period of time ϵ, there is approximately a $1 - 0.8\epsilon$ chance of being in state 1, a 0.32ϵ chance of being in state 2, and a 0.48ϵ chance of being in state

3. The approximation has error $O(\epsilon^2)$. Alternatively, it states that if the process is in state 1 it remains there for a duration exponentially distributed: $p(\text{dwell time} = t) = 0.8e^{-0.8t}$. And when it leaves, it will transition to state 2 with probability $0.32/0.8 = 0.4$ and to state 3 with probability $0.48/0.8 = 0.6$.

2.1.2 Racing Exponentials

We can also view a row of the matrix as describing racing exponential distributions. There are two important properties of an exponential distribution. First, it is memoryless:

$$p_Z(t) = p_Z(t+s|Z>s) \qquad \text{if } Z \text{ is exponentially distributed} \tag{6}$$

and thus is the right distribution for dwell times in a Markovian process. (The amount of time the process has stayed in this state does not affect the remaining dwell time.) It is also closed under minimization: Given a collection of random variables $Z_1, Z_2, \ldots, Z_n$,

$$p_{Z_i}(t) = r_i e^{-r_i t} \tag{7}$$

$$Y = \min_i Z_i \tag{8}$$

$$J = \arg\min_i Z_i \tag{9}$$

implies

$$p_Y(t) = re^{-rt} \tag{10}$$

$$\Pr(J = j) = \frac{r_j}{r} \tag{11}$$

where

$$r = \sum_{i=1}^{n} r_i \ . \tag{12}$$

That is, if we have a set of exponential distributions with (potentially) different rates, their minimum (the time of the earliest one) is also exponentially distributed with rate equal to the sum of the component rates. Furthermore, the component which causes this minimum is independent of the time and is proportional to that component's rate. Thus, we can view each row of the matrix as a set of "racing" exponential distributions: one for each potential next state with rates dictated by the off-diagonal elements. Whichever potential transition happens first is the one actually taken by the process, and the rest are discarded.

2.1.3 Event Decomposition

Further, we can use this to build an interpretation of the summation of two intensity matrices. If $\boldsymbol{Q} = \boldsymbol{Q_1} + \boldsymbol{Q_2}$, where both $\boldsymbol{Q_1}$ and $\boldsymbol{Q_2}$ are valid intensity matrices, then the process of $\boldsymbol{Q}$ can be viewed as the combination of processes of $\boldsymbol{Q_1}$ and $\boldsymbol{Q_2}$ in which both sub-processes "race" to produce the next transition: For each current state, the two sub-processes have their own rate of transition for the possible new states. Whichever transition happens first switches the state and the joint process continues with the new state. We can also view this as two different *event* types each with its own intensity matrix. The process of $\boldsymbol{Q}$ is the joint process of running both events at the same time, but throwing away (marginalizing out) the event types associated with the transitions, leaving only the transitions themselves.

2.1.4 Infinitesimal Rate Semantics

More formally, the dynamics of an n-state CTMP are described by a n-by-n intensity matrix $\boldsymbol{Q}$. The diagonal elements of $\boldsymbol{Q}$ are non-positive and the non-diagonal elements of $\boldsymbol{Q}$ are non-negative. The rows of $\boldsymbol{Q}$ sum to 0 (thus the diagonal elements are the negative row sums, if the diagonal element is excluded from the sum). We will denote the i, j element of $\boldsymbol{Q}$ as $q_{i,j}$. Further, for notational simplicity, we will let $q_i = -q_{i,i}$. That is, q_i is the rate of leaving state i, the absolute value of the corresponding diagonal element.

If we let $\boldsymbol{p}(t)$ be a row-vector of the marginal distribution of the process at time t, then the semantics of $\boldsymbol{Q}$ can be stated as

$$\boldsymbol{p}(t+\epsilon) = \boldsymbol{p}(t)(\boldsymbol{I} + \epsilon\boldsymbol{Q}) + o(\epsilon) \ . \tag{13}$$

This implies that

$$\begin{aligned}
\boldsymbol{p}(t+\epsilon) - \boldsymbol{p}(t) &= \epsilon\boldsymbol{p}(t)\boldsymbol{Q} + o(\epsilon) && (14)\\
\lim_{\epsilon\to 0}(\boldsymbol{p}(t+\epsilon) - \boldsymbol{p}(t))/\epsilon &= \boldsymbol{p}(t)\boldsymbol{Q} && (15)\\
\frac{d\boldsymbol{p}(t)}{dt} &= \boldsymbol{p}(t)\boldsymbol{Q} && (16)\\
& && (17)
\end{aligned}$$

This first-order linear homogeneous differential equation has solution

$$\boldsymbol{p}(t+s) = \boldsymbol{p}(t)e^{\boldsymbol{Q}s} \tag{18}$$

assuming $s > 0$ and the initial conditions at t, $\boldsymbol{p}(t)$, are known. The exponential is the matrix exponential, defined by its Taylor expansion:

$$e^{\boldsymbol{Q}s} = \sum_{k=0}^{\infty} \frac{s^k}{k!}\boldsymbol{Q}^k \ . \tag{19}$$

Although not often practical computationally, we can also express the matrix exponential in terms of the eigenvalues ($\{\lambda_i\}$) and corresponding right and left eigenvectors ($\{\boldsymbol{v}_i\}$ and $\{\boldsymbol{w}_i\}$) of $\boldsymbol{Q}$:

$$e^{\boldsymbol{Q}s} = \sum_i e^{\lambda_i s}\boldsymbol{v}_i\boldsymbol{w}_i^\top \ . \tag{20}$$

If $\boldsymbol{Q}$ is of finite size and irreducible (there is a positive-rate path from any state to any other state), then the process is ergodic (the notion of "cycling" behavior does not exist in continuous-time Markov processes) and there will be exactly one eigenvalue equal to 0. The corresponding right eigenvector is the unique steady-state (or stationary) distribution of the process. If the process is not ergodic, then there *may* be multiple 0 eigenvalues and no unique stationary distribution. All other eigenvalues will be less than 0 and correspond to transients in the system. Therefore, $\boldsymbol{Q}$ will always be negative semi-definite.

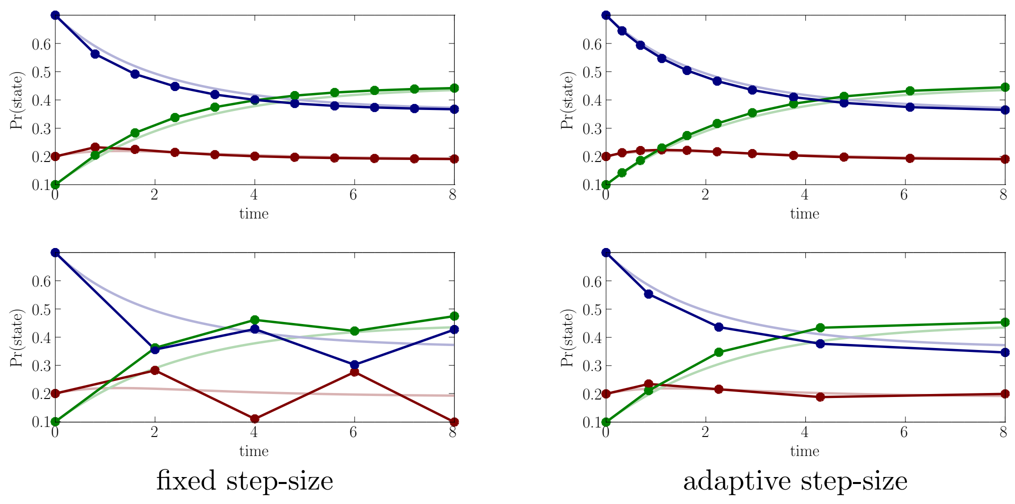


Figure 4: Propagation of marginal distribution from time 0 to time 8 by Euler integration. Left: fixed step-size. Right: adaptive step-size. Top: 11 evaluation points. Bottom: 5 evaluation points.

2.2 Matrix Exponential

The matrix exponential plays a critical role in many aspects of reasoning about continuous-time dynamic systems. At first, this would seem to be a significant downside, relative to discrete-time systems. Propagation of distribution $\boldsymbol{p}$ (as a vector) n time steps in a discrete-time system requires the multiplication by $\boldsymbol{M}^n$ (if $\boldsymbol{M}$ is the stochastic transition matrix). By contrast, the same operation in continuous-time requires the calculation of the matrix exponential, which is an infinite sum of matrix powers.

Consider computing the marginal distribution at time t by integrating the differential equation of Equation 18. The simplest method would be to use Euler integration with a fixed step size of Δt. This amounts to propagating over a fixed time interval by multiplying by $\boldsymbol{M} = \boldsymbol{I} + \Delta t \boldsymbol{Q}$. This is essentially the same as discretizing time and approximating the stochastic matrix over the resulting interval. To propagate to time t requires $t/\Delta t$ matrix multiplications. This is shown on the left side of Figure 4.

However, because time is continuous, we need not limit ourselves to time steps of uniform size. If we choose an adaptive step size, we can achieve the same accuracy with fewer evaluation points. Figure 4 demonstrates this for the simplest adaptive scheme (Euler steps with step size proportional to derivative magnitude). Note that for the same number of steps (computational complexity), the accuracy with adaptive steps is better.

For real applications, we would use a more advanced differential equation solver with more intelligent step size selection; see the work of Press, Teukolsky, Vetterling, and Flannery (1992) for an introduction. Yet, the idea is essentially the same: we can take larger steps during "less interesting" time periods. To do something similar with discrete time would require computations that essentially convert the discrete-time system to a continuous-time one. Techniques like squaring and scaling to multiply by large matrix powers can also be applied to the matrix exponential. For a full discussion of matrix exponential calculations, we refer to the excellent treatments of Moler and Loan (2003) and Najfeld and Havel (1994, 1995).

2.3 Uniformization

Uniformization (also called randomization) is a method for converting questions about a continuous-time Markov process into ones about a discrete-time one (Grassmann, 1977). Given an intensity matrix $\boldsymbol{Q}$, uniformization constructs the stochastic matrix

$$\boldsymbol{M} = \boldsymbol{Q}/\alpha + \boldsymbol{I} \tag{21}$$

where $\alpha \geq \max_i q_i$ (that is, α is no less than the largest rate in $\boldsymbol{Q}$). For example, the following uniformization is with $\alpha = 0.5$ (the smallest possible value for α).

$$\boldsymbol{Q} = \begin{bmatrix} -0.5 & 0.1 & 0.4 \\ 0.1 & -0.2 & 0.1 \\ 0.2 & 0.1 & -0.3 \end{bmatrix} \quad \to \quad \boldsymbol{M} = \begin{bmatrix} 0.0 & 0.2 & 0.8 \\ 0.2 & 0.6 & 0.2 \\ 0.4 & 0.2 & 0.4 \end{bmatrix} \tag{22}$$

The resulting stochastic matrix can be interpreted as a discrete-time process. However it is *not* equivalent to sampling the continuous-time process at uniform intervals. Nor is it, in general, equivalent to the embedded Markov chain (the sequence of states, discarding the times of transitions). The former is achieved through the matrix exponential and the latter is achieved by setting all diagonal elements of $\boldsymbol{Q}$ to zero and then normalizing each row to sum to one.

Rather, the discrete-time process associated with the stochastic matrix $\boldsymbol{M}$ is related to the continuous-time process associated with the intensity matrix $\boldsymbol{Q}$ in the following way. Consider the procedure of (1) sampling event times from a Poisson process with rate α (that is, the times between consecutive events are independently and identically distributed as an exponential distribution with rate α), then (2) sampling state transitions at these event times from the discrete-time process described by $\boldsymbol{M}$, and then (3) discarding any self transitions. This procedure produces samples from the same distribution as the original CTMP described by $\boldsymbol{Q}$.

This transformation is useful for simulation (sampling), but also for understanding and computing the matrix exponential. Because the intensity matrix $\boldsymbol{Q}$ is negative semi-definite, the Taylor expansion of the matrix exponential is unstable, as the "sign" of the terms alternates. However, we can fix this by using $\boldsymbol{M}$ instead of $\boldsymbol{Q}$. By reworking Equation 21, we note that $\boldsymbol{Q} = \alpha(\boldsymbol{M} - \boldsymbol{I})$. We can then write

$$e^{\boldsymbol{Q}t} = e^{\alpha(\boldsymbol{M}-\boldsymbol{I})t} \tag{23}$$

$$= e^{-\alpha t} e^{\alpha \boldsymbol{M} t} \qquad [e^{\boldsymbol{A}+\boldsymbol{B}} = e^{\boldsymbol{A}} e^{\boldsymbol{B}} \text{ if } \boldsymbol{AB} = \boldsymbol{BA}] \tag{24}$$

$$= e^{-\alpha t} \sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} \boldsymbol{M}^k \tag{25}$$

$$= \sum_{k=0}^{\infty} \underbrace{e^{-\alpha t} \frac{\alpha^k t^k}{k!}}_{\beta_k} \boldsymbol{M}^k \tag{26}$$

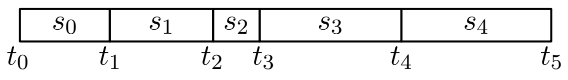


Figure 5: Pictorial representation of a finite-length sample from a CTMP.

where β_k is the probability of having exactly k events from a Poisson process with rate α in time t. This series is more stable ($\boldsymbol{M}$ is positive semi-definite) and for a finite number of terms, the sum is a quasi-probability vector (it is non-negative and sums to less than 1). The missing probability is a bound on the error. The sequence β_k grows and then decays. Therefore, discarding not only the tail of the series, but also early terms can speed up computations. Fox and Glynn (1988) give a method to compute left and right bounds on k to ensure a desired error tolerance.

Note that, if $\boldsymbol{Q}$ represents an ergodic continuous-time Markov process, then $\boldsymbol{M}$ represents an ergodic discrete-time Markov process when α is strictly greater than $\max_i q_i$ (a sufficient, but not necessary condition). If $\boldsymbol{M}$ is ergodic, then the stationary distribution of $\boldsymbol{Q}$ is the same as the stationary distribution of $\boldsymbol{M}$.

2.4 Likelihood

A complete finite-length sample (trajectory) $\mathcal{Tr}$ from a CTMP is sequence of states and the times of the transitions, plus an ending time: $\mathcal{Tr} = \{(s_0, t_0), (s_1, t_1), \ldots, (s_{n-1}, t_{n-1})\}, t_n$. Figure 5 shows a pictorial representation, for $n = 5$. If we use the convention that the process starts at time 0, then t_0 must be equal to 0.

The likelihood of this sample is the product of the conditional probabilities of each event (the starting state, each dwell duration, and each state transition):

$$p(\mathcal{Tr}) = \overbrace{\Pr(X(t_0) = s_0)}^{\text{init dist}} \prod_{i=0}^{n-2} \left(\overbrace{q_{s_i} e^{-q_{s_i}(t_{i+1}-t_i)}}^{\text{density of duration}} \overbrace{\frac{q_{s_i,s_{i+1}}}{q_{s_i}}}^{\text{pr of trans}} \right) \overbrace{e^{-q_{s_{n-1}}(t_n - t_{n-1})}}^{\text{pr of last duration}} \tag{27}$$

$$= P_0(s_0) \prod_{i=0}^{n-1} e^{-q_{s_i}(t_{i+1}-t_i)} \prod_{i=0}^{n-2} q_{s_i,s_{i+1}} \tag{28}$$

$$\ln p(\mathcal{Tr}) = \ln P_0(s_0) - \sum_{i=0}^{n-1} q_{s_i}(t_{i+1} - t_i) + \sum_{i=0}^{n-2} \ln q_{s_i,s_{i+1}} \tag{29}$$

We let P_0 be the distribution over the starting state of the process. Note that at time t_n the process does not transition. Rather, we observe that the process remains in state s_{n-1} for a duration of at least $t_n - t_{n-1}$.

Equation 29 can be rewritten as

$$\ln p(\mathcal{Tr}) = \ln P_0(s_0) - \sum_{s} T[s] q_s + \sum_{s \neq s'} N[s, s'] \ln q_{s,s'} \tag{30}$$

where $T[s]$ is the total time spent in state s and $N[s, s']$ is the total number of transitions from s to s', both of which are functions of $\mathcal{Tr}$. This demonstrates that a CTMP is a member of an exponential family in which the sufficient statistics are $T[\cdot]$ and $N[\cdot,\cdot]$ (plus the relevant sufficient statistics for the starting distribution), and the natural parameters are the diagonal elements of the intensity matrix and the logarithm of the non-diagonal elements. The likelihood of multiple trajectories has the same form, where $T[s]$ and $N[s,s']$ are the sums of the sufficient statistics over the individual trajectories.

2.4.1 Parameter Estimation

The maximum likelihood parameters can be easily derived by differentiating Equation 30, after replacing q_s with $\sum_{s'\neq s} q_{s,s'}$:

$$\frac{\partial \ln p(\mathcal{T}r)}{\partial q_{s,s'}} = \frac{N[s,s']}{q_{s,s'}} - T[s] \qquad \forall\ s' \neq s \tag{31}$$

which implies the ML parameters are

$$\hat{q}_{s,s'} = N[s,s']/T[s] \qquad \forall\ s' \neq s \tag{32}$$

$$\hat{q}_s = \sum_{s'\neq s} N[s,s']/T[s] \qquad \forall\ s \tag{33}$$

A maximum a posteriori (MAP) estimate can be calculated if we place suitable prior distributions on the parameters. In particular, we will put an independent gamma distribution prior over each of the independent parameters, $q_{s,s'}, \forall\ s \neq s'$:

$$p(q_{s,s'}; \alpha_{s,s'}, \tau_{s,s'}) = \frac{\tau_{s,s'}^{\alpha_{s,s'}+1}}{\Gamma(\alpha_{s,s'}+1)} q_{s,s'}^{\alpha_{s,s'}} e^{-q_{s,s'}\tau_{s,s'}} \tag{34}$$

which has parameters $\alpha_{s,s'}$ and $\tau_{s,s'}$. The posterior distribution over the parameters given data summarized by the sufficient statistics $T[s]$ and $N[s,s']$ is also gamma-distributed with parameters $\alpha_{s,s'} + N[s,s']$ and $\tau_{s,s'} + T[s]$. Thus, the MAP estimates of the parameters are

$$\hat{q}_s = \sum_{s'} \frac{N[s,s'] + \alpha_{s,s'}}{T[s] + \tau_{s,s'}} \tag{35}$$

$$\hat{q}_{s,s'} = \frac{N[s,s'] + \alpha_{s,s'}}{T[s] + \tau_{s,s'}}\ . \tag{36}$$

2.5 Inference

We will now consider two classic problems of reasoning in temporal systems: filtering and smoothing. Initially, we will assume we have observations (evidence) with a pattern like that in Figure 6: a sequence of times $\{t_0, t_1, \ldots, t_k\}$ and a sequence of evidences $\{e_1, e_2, \ldots, e_k\}$. We assume that we know the prior marginal distribution over $X(t_0)$, either from previous reasoning or because $t_0 = 0$.

Filtering is the task of computing $p(X(t) \mid e_1, e_2, \ldots, e_k)$ for $t \geq t_k$. If the evidence at each point is an observation of the state of the system, the Markovian property of the process makes inference trivial. Instead we will assume that e_i is only probabilistically related to $X(t_i)$ but independent of everything else given $X(t_i)$. (This is analogous to a discrete-time

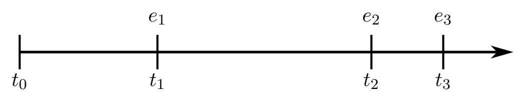


Figure 6: Example evidence pattern, point evidence.

hidden Markov model.) Thus we can view each observation as a noisy measurement of the system.

As with a hidden Markov model, we define a recursive filtering solution using a forward "message" $\boldsymbol{\alpha}$ whose components are defined as

$$\boldsymbol{\alpha}_i(t) = \Pr\left(X(t) = i, e_{[t_0,t)}\right) \tag{37}$$

where we denote $e_{[s,t)} = \{(t_i, e_i) \mid s \leq t_i < t\}$: the set of evidence in the interval $[s,t)$. By analogy we will also define $e_{[s,t]}$ and $e_{(s,t]}$ to be the evidence on $[s,t]$ and $(s,t]$ respectively (which we will need later). Note that $\boldsymbol{\alpha}$ is a *row* vector of probabilities, one for each state of the system. Recursive calculation of $\boldsymbol{\alpha}$ can be derived from

$$\Pr\left(X(t) = j, e_{[t_0,t)}\right) = \sum_i \Pr\left(X(s) = i, e_{[t_0,s)}\right) \Pr\left(X(t) = j, e_{[s,t)} \mid X(s) = i\right) \quad \forall\ t_0 \leq s < t \tag{38}$$

$$\boldsymbol{\alpha}(t) = \boldsymbol{\alpha}(s)\boldsymbol{F}(s,t) \qquad \forall\ t_0 \leq s < t \tag{39}$$

where the second equation is the vector version of the first equation, and the matrix $\boldsymbol{F}(s,t)$ has element i,j equal to $\Pr\left(X(t) = j, e_{[s,t)} \mid X(s) = i\right)$.

If there is no evidence in $[s,t)$, $\boldsymbol{F}(s,t) = e^{\boldsymbol{Q}(t-s)}$. Thus, we can propagate the distribution from one evidence time point to the next with the matrix exponential. To propagate across evidence times, we define

$$\boldsymbol{\alpha}^+(t) = \Pr\left(X(t), e_{[t_0,t]}\right) \tag{40}$$

to be the same as $\boldsymbol{\alpha}(t)$, but including evidence at t. If there is no evidence at t, the two vectors are the same. If there is evidence at t, then $\boldsymbol{\alpha}^+(t)$ is the same as $\boldsymbol{\alpha}(t)$, except that each element is multiplied by the probability of the evidence at that time point.

If we let $\boldsymbol{O^{(i)}}$ be a diagonal matrix in which diagonal element j is $\Pr(e_i \mid X(t_i) = j)$, then the recurrence for $\boldsymbol{\alpha}$ can be written as

$$\boldsymbol{\alpha}(t_0) = \boldsymbol{\alpha}^+(t_0) = \text{given} \tag{41}$$

$$\boldsymbol{\alpha}(t_i) = \boldsymbol{\alpha}^+(t_{i-1})e^{\boldsymbol{Q}(t_i - t_{i-1})} \qquad \forall\ 0 < i \leq k \tag{42}$$

$$\boldsymbol{\alpha}^+(t_i) = \boldsymbol{\alpha}(t_i)\boldsymbol{O^{(i)}} \qquad \forall\ 0 < i \leq k \tag{43}$$

$$\boldsymbol{\alpha}(t) = \boldsymbol{\alpha}^+(t_i)e^{\boldsymbol{Q}(t-t_i)} \qquad \forall\ t_i < t \leq t_{i+1} \text{ or } t_i < t, i = k \tag{44}$$

Equation 42 is a special case of Equation 44. It propagates from "just after" one evidence time until "just before" the next. Equation 43 propagates "across" an evidence point.

Equation 44 can be used to construct the filtered estimate at any non-evidence time by normalizing $\boldsymbol{\alpha}(t)$ to sum to 1 (dividing by the probability of the evidence prior to t).

Finally, note that a similar set of recurrences can be derived for $F(\cdot,\cdot)$. The result allows for the propagation of any distribution across time intervals which includes evidence; that is, we are not restricted to any particular initial condition, $\boldsymbol{\alpha}(t_0)$. However, if only a single $\boldsymbol{\alpha}$ is to be propagated, computing F first is more computationally expensive.

2.5.1 More Complex Evidence

The above filtering equations are just an inhomogeneous hidden Markov model (that is, the transition matrix is not constant) and familiar to those who have employed hidden Markov models. However, with continuous time, there are evidence patterns that do not have direct corresponding analogies in discrete-time. If the evidence consists of a finite number of observations, we can convert it into a similar form, by breaking time into intervals of constant evidence.

For instance, we might observe that the system is in a subset of states for a duration of time: During this time interval, the system does not leave the subset, but we do not observe whether there are any transitions within the subset. We can augment our evidence to include this information. For each interval $[t_{i-1}, t_i)$, we let S_i denote the subset of states in which the evidence constrains the system. If there are no such constraints, S_i is the full state space. For time points at which there is a change in S_i, but no point evidence, $\boldsymbol{O^{(i)}}$ is the identity matrix (inducing no change in the filtering estimate). Both S_i and $\boldsymbol{O^{(i)}}$ may be non-trivial for the same i.

Now to propagate from t_{i-1} to t_i, we must use a modified intensity matrix. In particular, we set to zero any rate which is inconsistent with the evidence S_i: all rates to, from, or within the set of states that are *not* in S_i. Let $\boldsymbol{Q^{(i)}}$ denote such a matrix. If the rows and columns are permuted such that the states in S_i are in the upper left corner, then this matrix has the form

$$\boldsymbol{Q^{(i)}} = \begin{bmatrix} \boldsymbol{\tilde{Q}_{S_i}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \tag{45}$$

where $\boldsymbol{\tilde{Q}_{S_i}}$ is the submatrix of $\boldsymbol{Q}$ of the rows and columns corresponding to S_i. Additionally, we modify $\boldsymbol{O^{(i-1)}}$, setting to 0 any diagonal elements corresponding to states *not* in S_i.

Note that $\boldsymbol{Q^{(i)}}$ is not (strictly) an intensity matrix: its rows do not sum to 0. In general, a diagonal element is greater (in absolute value) than the sum of the other row elements because we have set to zero non-diagonal rates. This "missing rate" corresponds to the probability of leaving the evidence set (and therefore not conforming to the evidence). While $e^{\boldsymbol{Q}(t_i - t_{i-1})}$ is a stochastic matrix representing the conditional distribution at time t_i given the state at time t_{i-1}, $e^{\boldsymbol{Q^{(i)}}(t_i - t_{i-1})}$ is a substochastic matrix (the row sums are less than or equal to 1), where the sum of each row is the probability of the evidence over the interval, given the state at time t_{i-1}.

The new filtering recurrence is

$$\begin{aligned}
\boldsymbol{\alpha}(t_0) &= \text{given} && && (46)\\
\boldsymbol{\alpha}(t_i) &= \boldsymbol{\alpha}^+(t_{i-1})e^{\boldsymbol{Q^{(i)}}(t_i-t_{i-1})} && \forall\ 0 < i \le k && (47)\\
\boldsymbol{\alpha}^+(t_i) &= \boldsymbol{\alpha}(t_i)\boldsymbol{O^{(i)}} && \forall\ 0 < i \le k && (48)\\
\boldsymbol{\alpha}(t) &= \boldsymbol{\alpha}^+(t_i)e^{\boldsymbol{Q^{(i+1)}}(t-t_i)} && \forall\ t_i < t \le t_{i+1} \text{ or } t_i < t, i = k\ . && (49)
\end{aligned}$$

We might also observe a transition at an exact time point. More generally, at time t_i we might observe that a transition occurred from one state of the set U_i^- to one state of the set U_i^+ (without knowing exactly which states within the sets). In this case, elements of $\boldsymbol{\alpha}(t)$ have the probabilities of a duration lasting until at least t, and $\boldsymbol{\alpha}^+(t)$ should have the probability density of a duration lasting exactly until t. The difference between the probability of the tail of an exponential and the density at the same point is just a multiplication by the relevant rate q. Thus, for this type of evidence, we can just modify $\boldsymbol{O^{(i)}}$. In particular,

$$\boldsymbol{O^{(i)}}_{j,k} = \begin{cases} q_{j,k} & \text{if } \ j \in U_i^-,\, k \in U_i^+, \text{ and } j \ne k \\ 0 & \text{otherwise} \end{cases} . \tag{50}$$

The recurrence remains the same, with the new definition of $\boldsymbol{O^{(i)}}$. Other evidence types are also possible and can be derived from the above types by augmenting the state space.

2.5.2 Smoothing

Smoothing is the problem of calculating $\Pr\left(X(t) \mid e_{[t_0,t_k]}\right)$ for $t_0 \le t \le t_k$. As common with Markov processes, we note that

$$\Pr\left(X(t) \mid e_{[t_0,t_k]}\right) \propto \Pr\left(X(t) \mid e_{[t_0,t)}\right) \Pr\left(e_{[t,t_k]} \mid X(t)\right) \tag{51}$$

where the constant of proportionality can be found by noting that the sum of Equation 51 over the value of $X(t)$ must equal 1. The first term on the right is calculated with the $\boldsymbol{\alpha}(\cdot)$ recurrence above. The second term we calculate with a backward message recurrence. Define

$$\boldsymbol{\beta}_i(t) = \Pr\left(e_{[t,t_k]} \mid X(t) = i\right) \tag{52}$$
$$\boldsymbol{\beta}_i^+(t) = \Pr\left(e_{(t,t_k]} \mid X(t) = i\right) \tag{53}$$

If we let $\boldsymbol{\beta}$ be a *column* vector, then the backward recurrence is analogous the forward one, but with right multiplication instead of left multiplication:

$$\begin{aligned}
\boldsymbol{\beta}^+(t_k) &= \mathbf{1} && \text{vector of 1s} && (54)\\
\boldsymbol{\beta}(t_i) &= \boldsymbol{O^{(i)}}\boldsymbol{\beta}^+(t_i) && \forall\ 0 < i \le k && (55)\\
\boldsymbol{\beta}^+(t_{i-1}) &= e^{\boldsymbol{Q^{(i)}}(t_i-t_{i-1})}\boldsymbol{\beta}(t_i) && \forall\ 0 < i \le k && (56)\\
\boldsymbol{\beta}(t) &= e^{\boldsymbol{Q^{(i)}}(t_i-t)}\boldsymbol{\beta}(t_i) && \forall\ t_{i-1} \le t < t_i\ . && (57)
\end{aligned}$$

For any time t, the vector of the distribution of the state of the system at t given all the evidence is

$$\boldsymbol{p}(X(t) \mid e_{[t_0,t_k]}) \propto \boldsymbol{\alpha}(t) \odot \boldsymbol{\beta}(t) \tag{58}$$

where $\odot$ is the Hadamard (point-wise) product.

2.6 Parameter Estimation from Incomplete Evidence

Section 2.4.1 demonstrated that a CTMP is a member of an exponential family with sufficient statistics $T[i]$ (the amount of time spent in state i) and $N[i,j]$ (the number of transitions from i to j). If the evidence trajectories are fully observed over a continuous interval of time, then these sufficient statistics can be trivially tallied. Further, if each evidence trajectory is observed at $t = 0$, the sufficient statistics for the initial distribution are also directly observed.

However, if portions of the interval are hidden, or more generally the observations are of the form of the previous section, direct likelihood maximization is not feasible. There are two basic approaches for maximum likelihood estimation in this case: gradient ascent and expectation maximization (EM).

For gradient ascent, we can replace the sufficient statistics in Equation 31 with their expected values The standard argument for exponential models applies: Let $\mathcal{Tr}$ be a partially observed trajectory and let h stand for any potential completion of it.

$$\ln p(\mathcal{Tr}) = \ln \sum_h e^{\ln p(\mathcal{Tr},h)} \tag{59}$$

$$\frac{\partial \ln p(\mathcal{Tr})}{\partial q_{i,j}} = \frac{1}{p(\mathcal{Tr})} \sum_h p(\mathcal{Tr},h) \frac{\partial \ln p(\mathcal{Tr},h)}{\partial q_{i,j}} \tag{60}$$

$$= E_{h|\mathcal{Tr}} \left[\frac{N[i,j]}{q_{i,j}} - T[i] \right] \tag{61}$$

$$= \left(\frac{\bar{N}[i,j]}{q_{i,j}} - \bar{T}[i] \right) \tag{62}$$

where $\bar{N}[i,j]$ and $\bar{T}[i]$ are the expected values of $N[i,j]$ and $T[i]$ with respect to completions of $\mathcal{Tr}$. For EM, we similarly replace $N[i,j]$ and $T[i]$ in Equation 32 with $\bar{N}[i,j]$ and $\bar{T}[i]$. We are therefore left with the problem of computing the expected values of $N[i,j]$ and $T[i]$.

Full derivations are shown in the work of Nodelman et al. (2003). A quick version for $\bar{T}[i]$ is

$$\bar{T}[i] = \int_{t_0}^{t_k} p(X(t) = i \mid e_{[t_0,t_k]})\; dt \tag{63}$$

$$= \frac{1}{p(\mathcal{Tr})} \int_{t_0}^{t_k} \boldsymbol{\alpha}_i(t) \boldsymbol{\beta}_i(t)\; dt\ . \tag{64}$$

The expected value of $N[i,j]$ has a similar form:

$$\bar{N}[i,j] = \frac{q_{i,j}}{p(\mathcal{Tr})} \int_{t_0}^{t_k} \boldsymbol{\alpha}_i(t) \boldsymbol{\beta}_j(t)\; dt + \sum_{l \in \text{Trans}} \frac{\boldsymbol{\alpha}_i(t_l) \boldsymbol{O^{(l)}}_{i,j} \boldsymbol{\beta}_j^+(t_l)}{\sum_{i',j'} \boldsymbol{O^{(l)}}_{i',j'}} \tag{65}$$

where Trans is the set of evidence indices at which time a transition was (perhaps partially) observed: The first term handles unobserved transitions and the second handles (partially) observed transitions.

If we let $\boldsymbol{\Delta_{i,j}}$ be a matrix of all zeros, except for a single one in location (i,j), the integrals in both Equation 64 and Equation 65 have the form

$$\int_{t_0}^{t_k} \boldsymbol{\alpha}_i(t)\boldsymbol{\beta}_j(t)\ dt = \sum_{l=1}^{k} \int_{t_{l-1}}^{t_l} \boldsymbol{\alpha}_i(t)\boldsymbol{\beta}_j(t)\ dt \tag{66}$$

$$= \sum_{l=1}^{k} \int_{t_{l-1}}^{t_l} \boldsymbol{\alpha}^+(t_{l-1})e^{\boldsymbol{Q^{(i)}}(t-t_{l-1})}\boldsymbol{\Delta_{i,j}}e^{\boldsymbol{Q^{(i)}}(t_l-t)}\boldsymbol{\beta}(t_l)\ dt\ . \tag{67}$$

Thus, after a forward and backward pass to calculate $\boldsymbol{\alpha}^+(i)$ and $\boldsymbol{\beta}(i)$ at each evidence change point i, each integral is relatively simple. They can be solved by standard quadrature methods or by the solution of a differential equation (Asmussen, Nerman, & Olsson, 1996). Alternatively, the calculation of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ usually results in values for each at various time points, which can be interpolated to full functions and used to directly solve the integrals.

3. Kronecker Algebra Representations

When the number of states is no more than a few thousand, the above methods are computationally feasible on a modern computer. However, most models are described in terms of assignments to variables. Thus the number of states grows exponentially with the number of variables. For more than a few tens of variables, we must seek more compact representations.

For the remainder of this paper, we will consider the state space of the process X to be an assignment to L variables, $\{X_1, X_2, \ldots, X_L\}$. We let variable X_i have n_i possible assignments. Thus, the total state space is of size $n = \prod_{i=1}^{L} n_l$. We let a bold $\boldsymbol{x}$ stand for a state (joint assignment to all L variables), with component x_i being the assignment to variable i in state $\boldsymbol{x}$. Such a state space is often referred to as *factored* or *structured* or *variable-based*.

Kronecker products and sums are natural "basic operations" from which to build compact representations of the process intensities. In some cases, these compact representations naturally describe the transition rates, but do not as naturally describe the diagonal elements of $\boldsymbol{Q}$ (the negative rates of leaving each state). Thus, we will define $\boldsymbol{R}$ to be the same as $\boldsymbol{Q}$, except with zeros at each diagonal position. The diagonals can be reconstructed from the non-diagonal elements in the same row, so the information content is the same.

3.1 Kronecker Product

The first basic operation is the Kronecker product. Given matrices $\boldsymbol{A^{(1)}}, \boldsymbol{A^{(2)}}, \ldots, \boldsymbol{A^{(K)}}$ where $\boldsymbol{A^{(k)}}$ is of general size m_k-by-n_k, the Kronecker product is written

$$\boldsymbol{A} = \bigotimes_{k=1}^{K} \boldsymbol{A^{(k)}} \tag{68}$$

where $\boldsymbol{A}$ (the result) is an m-by-n matrix: $m = \prod_k m_k$ and $n = \prod_k n_k$. The elements of $\boldsymbol{A}$ represent all possible multiplications of one element from each of $\boldsymbol{A^{(1)}}, \boldsymbol{A^{(2)}}, \ldots, \boldsymbol{A^{(K)}}$. Let $\mathcal{M}_k = \{1, 2, \ldots, m_k\}$ and $\mathcal{N}_k = \{1, 2, \ldots, n_k\}$, that is the valid indices into matrix $\boldsymbol{A^{(k)}}$.

$$\text{Given } \boldsymbol{A} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \quad \text{and} \quad \boldsymbol{B} = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$\boldsymbol{A} \otimes \boldsymbol{B} = \left[\begin{array}{c|c} a_{00}\boldsymbol{B} & a_{01}\boldsymbol{B} \\ \hline a_{10}\boldsymbol{B} & a_{11}\boldsymbol{B} \end{array}\right] = \left[\begin{array}{ccc|ccc} a_{00}b_{00} & a_{00}b_{01} & a_{00}b_{02} & a_{01}b_{00} & a_{01}b_{01} & a_{01}b_{02} \\ a_{00}b_{10} & a_{00}b_{11} & a_{00}b_{12} & a_{01}b_{10} & a_{01}b_{11} & a_{01}b_{12} \\ a_{00}b_{20} & a_{00}b_{21} & a_{00}b_{22} & a_{01}b_{20} & a_{01}b_{21} & a_{01}b_{22} \\ \hline a_{10}b_{00} & a_{10}b_{01} & a_{10}b_{02} & a_{11}b_{00} & a_{11}b_{01} & a_{11}b_{02} \\ a_{10}b_{10} & a_{10}b_{11} & a_{10}b_{12} & a_{11}b_{10} & a_{11}b_{11} & a_{11}b_{12} \\ a_{10}b_{20} & a_{10}b_{21} & a_{10}b_{22} & a_{11}b_{20} & a_{11}b_{21} & a_{11}b_{22} \end{array}\right]$$

Figure 7: Example Kronecker product

Then, let $\mathcal{I}_r$ be a mapping from $\mathcal{M}_1 \times \mathcal{M}_2 \times \cdots \times \mathcal{M}_K$ to $\{1, 2, \ldots, m\}$ and let $\mathcal{I}_c$ be similarly defined as a mapping from $\mathcal{N}_1 \times \mathcal{N}_2 \times \cdots \times \mathcal{N}_K$ to $\{1, 2, \ldots, n\}$. It does not matter usually what the mappings are, but by convention we take them to be lexicographic orderings (or mixed-base numbering index). For instance $\mathcal{I}_r(i_1, i_2, \ldots, i_K) = \sum_{1 \le k \le K} i_k m_{1:k-1}$ where $m_{a:b} = \prod_{a \le k \le b} m_k$. Then

$$\boldsymbol{A}_{\mathcal{I}_r(i_1,i_2,\ldots,i_K),\mathcal{I}_c(j_1,j_2,\ldots,j_K)} = \prod_{k=1}^{K} \boldsymbol{A^{(k)}}_{i_k,j_k} \ . \tag{69}$$

While the notation makes it appear complex, the concept is simple. Figure 7 demonstrates a simple example. In terms of sparsity (one measure of structure), the Kronecker product has a number of non-zero elements equal to the product of the number of non-zero elements in each input matrix.

A Kronecker product is analogous to a factor product (in Bayesian network terminology) if we treat each operand matrix as a factor over two different variables (and no matrices share the same variables), and the result matrix is a factor in which half of the variables are flattened into the "column" dimension and the other half are flattened into the "row" dimension.

In terms of distributions, the Kronecker product represents *independence*. Given two variables X_1 and X_2 with marginal distributions represented by the vectors $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, $\boldsymbol{v_1} \otimes \boldsymbol{v_2}$ is a joint distribution over both X_1 and X_2. In particular, it is the independent joint distribution with marginals $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$.

In terms of a rate matrix, the Kronecker product represents *synchronization* (Plateau, 1985). If we have two variables, X_1 and X_2 with rate[2] matrices $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$, $\boldsymbol{R_1} \otimes \boldsymbol{R_2}$ is a rate matrix over the state space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$ (joint assignments to X_1 and X_2). It represents a rate matrix in which changes in the state of X_1 must occur at the same time as those in the state of X_2 (both variables will be changed by every transition).

2. This does not hold generally for intensity matrices, as the Kronecker product does not do anything sensible with the diagonal elements.

$$\boldsymbol{A} \oplus \boldsymbol{B} = \boldsymbol{A} \otimes \boldsymbol{I_3} + \boldsymbol{I_2} \otimes \boldsymbol{B} =$$

$$\left[\begin{array}{ccc|ccc}
a_{0,0} & & & a_{0,1} & & \\
& a_{0,0} & & & a_{0,1} & \\
& & a_{0,0} & & & a_{0,1} \\
\hline
a_{1,0} & & & a_{1,1} & & \\
& a_{1,0} & & & a_{1,1} & \\
& & a_{1,0} & & & a_{1,1}
\end{array}\right] + \left[\begin{array}{ccc|ccc}
b_{0,0} & b_{0,1} & b_{0,2} & & & \\
b_{1,0} & b_{1,1} & b_{1,2} & & & \\
b_{2,0} & b_{2,1} & b_{2,2} & & & \\
\hline
& & & b_{0,0} & b_{0,1} & b_{0,2} \\
& & & b_{1,0} & b_{1,1} & b_{1,2} \\
& & & b_{2,0} & b_{2,1} & b_{2,2}
\end{array}\right] =$$

$$\left[\begin{array}{ccc|ccc}
a_{0,0}+b_{0,0} & b_{0,1} & b_{0,2} & a_{0,1} & & \\
b_{1,0} & a_{0,0}+b_{1,1} & b_{1,2} & & a_{0,1} & \\
b_{2,0} & b_{2,1} & a_{0,0}+b_{2,2} & & & a_{0,1} \\
\hline
a_{1,0} & & & a_{1,1}+b_{0,0} & b_{0,1} & b_{0,2} \\
& a_{1,0} & & b_{1,0} & a_{1,1}+b_{1,1} & b_{1,2} \\
& & a_{1,0} & b_{2,0} & b_{2,1} & a_{1,1}+b_{2,2}
\end{array}\right]$$

Figure 8: Example Kronecker sum, given same matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as in Figure 7. Zeros are omitted. Note that the non-zero off-diagonal entries all correspond to only one of the two indices (into $\boldsymbol{A}$ or $\boldsymbol{B}$) changing.

3.2 Kronecker Sum

The other Kronecker operation is the Kronecker sum. It is only defined on square matrices. Given square matrices $\boldsymbol{A^{(1)}}, \boldsymbol{A^{(2)}}, \ldots, \boldsymbol{A^{(K)}}$ where $\boldsymbol{A^{(k)}}$ has size n_k-by-n_k, the Kronecker sum is defined in terms of the Kronecker product:

$$\boldsymbol{A} = \bigoplus_{k=1}^{K} \boldsymbol{A^{(k)}} = \sum_{k=1}^{K} \boldsymbol{I_{n_1}} \otimes \boldsymbol{I_{n_2}} \otimes \ldots \boldsymbol{I_{n_{k-1}}} \otimes \boldsymbol{A^{(k)}} \otimes \boldsymbol{I_{n_{k+1}}} \otimes \cdots \otimes \boldsymbol{I_{n_K}} \tag{70}$$

where $\boldsymbol{I_n}$ is the identity matrix of size n-by-n. The Kronecker sum has the same size as the Kronecker product of the same matrices and we can use the same indexing function to reference elements in the sum, but we need only one because the matrix is square, thus $\mathcal{I}_r = \mathcal{I}_c = \mathcal{I}$:

$$A_{\mathcal{I}(i_1,i_2,\ldots,i_K),\mathcal{I}(j_1,j_2,\ldots,j_K)} = \begin{cases} \sum_{k=1}^{K} \boldsymbol{A^{(k)}}_{i_k,i_k} & \text{if } i_l = j_l \text{ for all } l \\ \boldsymbol{A^{(k)}}_{i_k,j_k} & \text{if } i_l = j_l \text{ for all } l \text{ except } l = k \\ 0 & \text{otherwise.} \end{cases} \tag{71}$$

Figure 8 demonstrates a simple example.

In terms of a CTMP, the Kronecker sum represents asynchronicity. Given two variables, X_1 and X_2 with intensity[3] matrices $\boldsymbol{Q_1}$ and $\boldsymbol{Q_2}$, $\boldsymbol{Q_1} \oplus \boldsymbol{Q_2}$ is an intensity matrix over the joint state space in which each process's events proceed irrespective of the other's state. That is, the processes are independent (assuming their starting distributions are independent).

3. The same holds for rate matrices, but we can be stronger here than for the Kronecker product and make this statement about the intensity matrices too.

Note that the intensity of any transition that involves two or more variables is zero (at any instant, a maximum of one variable can change).

3.3 Properties

The Kronecker product obeys the classic distributive property:

$$(\boldsymbol{A}+\boldsymbol{B})\otimes\boldsymbol{C}=\boldsymbol{A}\otimes\boldsymbol{C}+\boldsymbol{B}\otimes\boldsymbol{C}$$

The mixed product property provides a relationship between the Kronecker product and the matrix product. Given matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ and assuming that $\boldsymbol{AC}$ and $\boldsymbol{BD}$ are valid matrix products,

$$(\boldsymbol{A}\otimes\boldsymbol{B})(\boldsymbol{C}\otimes\boldsymbol{D})=(\boldsymbol{AC})\otimes(\boldsymbol{BD})\ . \tag{72}$$

One consequence is that the Kronecker product can be expressed as

$$\bigotimes_{k=1}^{K}\boldsymbol{A^{(k)}}=\prod_{k=1}^{K}\boldsymbol{I_{n_1}}\otimes\boldsymbol{I_{n_2}}\otimes\cdots\otimes\boldsymbol{I_{n_{k-1}}}\otimes\boldsymbol{A^{(k)}}\otimes\boldsymbol{I_{n_{k+1}}}\otimes\cdots\otimes\boldsymbol{I_{n_K}} \tag{73}$$

$$=\prod_{k=1}^{K}\boldsymbol{I_{n_{1:k-1}}}\otimes\boldsymbol{A^{(k)}}\otimes\boldsymbol{I_{n_{k+1:K}}} \tag{74}$$

where $n_{a:b}$ is the product of the terms n_a through n_b as defined above. This shows a bit of the relationship between Kronecker products and sums: Compare Equation 70 and Equation 73.

This can be further reworked as

$$\bigotimes_{i=k}^{K}\boldsymbol{A^{(k)}}=\prod_{i=k}^{K}\boldsymbol{P_{n_{1:k},n_{k+1:K}}}^{\top}\cdot(\boldsymbol{I_{\overline{n}_k}}\otimes\boldsymbol{A^{(k)}})\cdot\boldsymbol{P_{n_{1:k},n_{k+1:K}}} \tag{75}$$

where $\overline{n}_k=n_{1:K}/n_k$ and $\boldsymbol{P_{a,b}}$ is the matrix describing an a,b-perfect shuffle permutation of $(0,...,ab-1)$: its entry in position (i,j) is 1 if $j=(i \bmod a)\cdot b+\lfloor i/a\rfloor$, and 0 otherwise (in particular, $\boldsymbol{P_{a,b}}=\boldsymbol{P_{a,b}}=\boldsymbol{I_{a\cdot b}}$ if a or b is 1). Whereas Equation 74 orders the Kronecker products of the outer product's terms so that the elements of $\boldsymbol{A_k}$ are in the correct places, Equation 75 repeats $\boldsymbol{A_k}$ on the diagonal and then permutes the rows and columns to place the elements in the correct locations. A similar transformation can be used to rewrite Equation 70 as a sum of shuffled block-diagonal matrices. Because the permutations can often be done implicitly in code, these versions can be useful in deriving algorithms.

3.4 Compact Kronecker Representations

Given a factored state space as before, any joint rate matrix $\boldsymbol{R}$ can be expressed as a sum of Kronecker products:

$$\boldsymbol{R}=\sum_{e=1}^{E}\bigotimes_{l=1}^{L}\boldsymbol{R_e^{(l)}} \tag{76}$$

where there are L variables and $\boldsymbol{R_e^{(l)}}$ is a rate matrix over the space of variable l only. In particular, an exponentially sized (in the number of variables) representation is straightforward: e ranges over the elements in the resulting matrix. For element corresponding to

$(x_1, x_2, \ldots, x_L), (x'_1, x'_2, \ldots, x'_L)$, $\boldsymbol{R}_{\boldsymbol{e}}^{(\boldsymbol{l})} = \boldsymbol{\Delta}_{\boldsymbol{x_l},\boldsymbol{x'_l}}$ for $1 < l \leq L$ and the same for $l = 1$, except multiplied by the scalar value to be placed in this location. In this way each term in the sum is a matrix with at most a single non-zero element. However, for many processes we can expect E to be a manageable number. For instance, if the variables are all independent, $E = L$ (and all but L of the L^2 rate components are identity matrices), as per the Kronecker sum above.

We can view each of the E terms in the sum as separate "events" whose identities have been marginalized out to produce the resulting process (see Section 2.1.3). These events must couple variables synchronously (due to the Kronecker product). We exploit this type of decomposition more extensively in the next sections.

4. Decision Diagram Representations

While the encoding in Equation 76 can be efficient, we can do better by exploiting more internal structure. $\boldsymbol{R}$ can be viewed as a mapping from two discrete domains (the row index and the column index) to a real value. *Decision diagrams* have long been used in computer science to compactly encode functions over discrete domains. Here we show how they have been used in CTMPs and how they can be seen as an alternative to Kronecker algebra encodings, in the case of the MTBDDs used in PRISM (Kwiatkowska, Norman, & Parker, 2011), or even as an extension of Kronecker algebra encodings, in the case of the Matrix Diagrams used in Möbius (Deavours, Clark, Courtney, Daly, Derisavi, Doyle, Sanders, & Webster, 2002) or the EV*MDDs used in SMART (Ciardo, Jones, Miner, & Siminiceanu, 2006).

4.1 Decision Diagram Overview

Decision diagrams encode functions of the form $f : X \to X_0$ where, as before, the *domain* state space X is structured: $X = X_1 \times \cdots \times X_L$. In other words, f is applied to a (state) tuple and evaluates to an element of a *range* set X_0. One can then think of f as the encoding of a vector indexed by X and having entries with values in X_0. Of course, the same idea can be employed to encode matrices, we simply need to use the domain $X \times X$. (In practice, we actually use the *interleaved* domain $X'_1 \times X_1 \times \cdots \times X'_L \times X_L$, where the "unprimed" state variables refer to row indices, or "from" states, while the "primed" state variable refer to column indices, or "to" states, as this usually leads to more compact decision diagrams.)

Binary decision diagrams, or BDDs (Bryant, 1986), encode functions for which all sets forming the domain X are binary, while multiway decision diagrams, or MDDs (Kam, Villa, Brayton, & Sangiovanni-Vincentelli, 1998), allow non-binary domain sets. However, for both the range X_0 is binary. For our numeric application, we need to extend such representations to allow the range X_0 to be either the integers $\mathbb{Z}$ (possibly augmented with the value ∞ to indicate "undefined") or the reals $\mathbb{R}$ (possibly, again, augmented with ∞, or restricted to the nonnegative reals $\mathbb{R}^{\geq 0}$). The range $\mathbb{Z}$ is used primarily to encode indexing functions for non-consecutive sets of states. The range $\mathbb{R}$ is used to encode the rates themselves.

Informally, decision diagrams are directed acyclic graphs organized in layers with each layer corresponding to a different variable in the domain of the function. The outgoing edges from a node correspond to the values the variable on that layer can take on. The value of the function is determined by following the path from the root corresponding to the values taken by the domain variables. The path ends in a terminal node which, in BDDs and MDDs, give the value of the function.

A first proposal to encode non-binary function was to extend BDDs and MDDs so that, instead of the terminals 0 and 1, any element of X_0 can be a terminal node. The resulting *multi-terminal* (Clarke, Fujita, McGeer, Yang, & Zhao, 1993) BDDs (or MTMDDs) are quite general. However, as we will see, MTMDDs are sometimes unable to compactly encode even simple functions. We therefore focus on a newer class of *edge-valued* decision diagrams, which can be exponentially more compact, and provably never larger, than MTMDDs (Roux & Siminiceanu, 2010). For edge-valued decision diagrams, a value is associated with each edge in the tree, and the function's value is determined from the values along the path to the terminal node. The exact definition of these diagrams depends on the operator used to combine edge values. We consider two cases, EV$^+$MDDs (Ciardo & Siminiceanu, 2002) (where X_0 is either $\mathbb{Z} \cup \{\infty\}$ or $\mathbb{R} \cup \{\infty\}$ and edge values along a path are summed) and EV*MDDs (Wan, Ciardo, & Miner, 2011) (where X_0 is $\mathbb{R}^{\geq 0}$ and edge values along a path are multiplied).

4.2 Multiterminal and Edge-Valued Decision Diagrams

Formally, both an EV$^+$MDD and an EV*MDD are acyclic directed edge-labeled and edge-valued graphs. A node in the graph p has a level $p.lvl$ and a set of directed edges indexed by x. The edge associated with label x is written as $p[x] = \langle p[x].val, p[x].ch \rangle$, where $p[x].val$ is the value associated with the edge and $p[x].ch$ is the target of the edge.

- The only terminal node (one without outgoing edges) is Ω, at level 0: $\Omega.lvl = 0$.
- A nonterminal node p is at level $k \in \{1, \ldots, L\}$: $p.lvl = k$. For each $x_k \in X_k$, it has an outgoing edge labeled x_k, associated with a value $v \in X_0$, and pointing to a node q with $q.lvl < k$. Thus $p[x_k] = \langle v, q \rangle$.
- A node p at level k encodes the function $f_p : X_1 \times \cdots \times X_k \to X_0$. For EV$^+$MDDs, f_p is defined recursively as $f_p = 0$ if $p = \Omega$, and $f_p(x_1, \ldots, x_k) = p[x_k].val + f_{p[x_k].ch}(x_1, \ldots, x_{p[x_k].ch.lvl})$ otherwise (that is, if p is a nonterminal node).

  For EV*MDDs, f_p is defined recursively as $f_p = 1$ if $p = \Omega$, and $f_p(x_1, \ldots, x_k) = p[x_k].val \cdot f_{p[x_k].ch}(x_1, \ldots, x_{p[x_k].ch.lvl})$ otherwise.

Most decision diagram definitions have additional restrictions to ensure *canonicity*, that is so that any representable function has a unique representation. For the edge-valued decision diagrams we have defined, this is achieved by additionally requiring all of the following.

- There are no duplicate nodes: if $p.lvl = q.lvl = k$ and, for each $x_k \in X_k$, we have $p[x_k] = q[x_k]$, then $p = q$.
- The *absorbing value* terminates a path: for EV$^+$MDDs, $p[x_k].val = \infty$ implies that $p[x_k].ch = \Omega$; for EV*MDDs, $p[x_k].val = 0$ implies that $p[x_k].ch = \Omega$.
- Each node p at level $k > 0$ is *normalized*: for EV$^+$MDDs, $\min\{p[x_k].val : x_k \in X_k\} = 0$; for EV*MDDs, $\max\{p[x_k].val : x_k \in X_k\} = 1$.

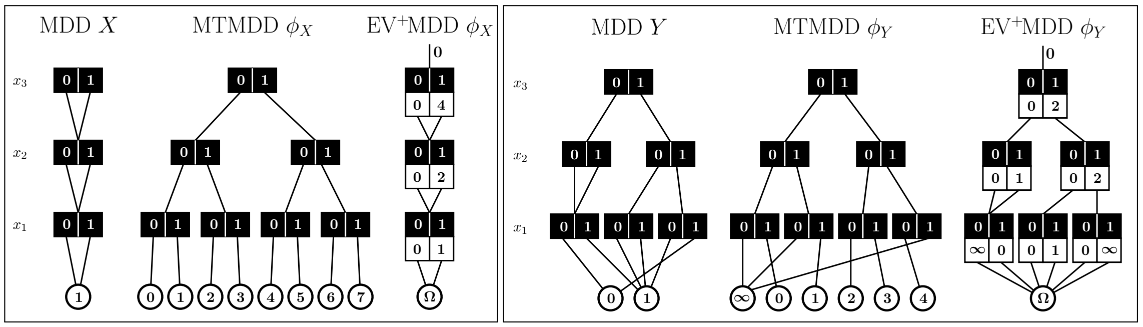


Figure 9: Encoding the lexicographic index function, ϕ_X, for set X. The left panel shows the quasi-reduced MDD encoding X followed by the MTMDD and the EV$^+$MDD encoding ϕ_X; the right panel shows the corresponding encodings for the set $Y = \{100, 110, 001, 101, 011\}$. In either case, the EV$^+$MDD is isomorphic to the MDD. Each level of the tree corresponds to a different variable. Black boxes are the values of this variable (and traversal of the diagram follows the edge leading out of this box for the value of input). White boxes (for EV$^+$MDD) are the values of the corresponding edge (which are summed to produce the function's value). The 0 at the top of the EV$^+$MDD is the value added to any path or traversal of the diagram.

Furthermore, we require that one of the following two *reduction forms* must be used.

- In *quasi-reduced* form, only nodes at level L have no incoming edges (except the special case of the graph consisting of just Ω) and the children of a node at level k are at level $k-1$ (except for absorbing-valued edges, which point to Ω, as stated above).

- In *fully-reduced* form, there are no redundant nodes, where a node p at level k is redundant if $p[x_k] = p[y_k]$ for all $x_k, y_k \in X_k$.

Strictly speaking, an EV$^+$MDD node encodes a function with values between 0 (included) and ∞ (possibly included); thus, a function f with range $\mathbb{Z} \cup \{\infty\}$ or $\mathbb{R} \cup \{\infty\}$ is encoded by $\langle\sigma,p\rangle$ where $\sigma = \min\{f(i) : i \in X\}$ and $f_p = f - \sigma$ (the special case $f \equiv \infty$ is encoded by the pair $\langle\infty,\Omega\rangle$). Analogously, an EV*MDD node encodes a function with values between 0 (possibly included) and 1 (included); thus a function f with range $\mathbb{R}^{\geq 0}$ is encoded by $\langle\sigma,p\rangle$ where $\sigma = \max\{f(i) : i \in X\}$ and $f_p = f/\sigma$ (the special case $f \equiv 0$ is encoded by the pair $\langle 0,\Omega\rangle$). In the following, we use the term EV$^+$MDD or EV*MDD also for the pair $\langle\sigma,p\rangle$, with the understanding that σ is just a parameter that scales the values of the function encoded by node p.

4.3 Lexicographic Index Example

We illustrate the compactness of these decision diagrams using the *lexicographic index*, also called the *mixed-base value*, of a state $x = (x_1, \ldots, x_L)$, defined as $\phi(x) = \sum_{1 \leq k \leq L} x_k \cdot n_{1:k-1}$, where $n_{a:b} = n_a \cdots n_b$ for $a \leq b$ (as in Section 3.1). We discuss the importance of this function after showing its encoding.

Figure 9 (left) shows the lexicographic index function ϕ (along side the MDD encoding the set of states). The MTMDD for ϕ is a full L-level tree with $n_{1:L}$ leaves. By contrast, the EV$^+$MDD for ϕ contains just one node at each level, where the child labeled x_k of the node at level k points to the node at level $k-1$ and has value $x_k \cdot n_{1:k-1}$.

Interestingly, this function retains a compact encoding even if we modify it so that it applies to a set $Y \subset X$, that is $\phi_Y(x) = |\{y \in Y : \phi(y) < \phi(x)\}|$ if $x \in Y$, and $\phi_Y(x) = \infty$ otherwise, in the sense that the MDD encoding Y and the EV$^+$MDD encoding ϕ_Y are isomorphic (right panel in Figure 9). This is of particular importance for the exact numerical solution of structured CTMPs whose reachable state space X_{rch} is a strict (and possibly complicated) subset of X, since in this case we need to frequently and efficiently map a state $x = (x_1, \ldots, x_L)$ to its index $\phi_{X_{rch}}(x)$ in a full probability vector of size $|X_{rch}|$. Compactly representing this index function is key to efficient calculations in such CTMPs.

Obviously, EV*MDDs can also be exponentially more compact than MTMDDs; simply consider that the EV*MDD encoding of $e^{-\phi}$ also has one node per level, where the child with label x_k of the node at level k has value $e^{-x_k \cdot n_{1:k-1}}$.

4.4 Decision Diagram Operations

In addition to efficiently *encoding* structured functions, decision diagrams are also able to efficiently *manipulate* functions. All decision diagram operations proceed recursively from the root node(s) and make extensive use of dynamic programming. Specifically, they use an *operation cache* to retrieve the result of a specific operation on a specific choice of parameters, if this result has been previously computed when exploring different paths in the recursion. This reduces the worst-case complexity of an operation (for example, computing $c = a + b$, where a and b are functions encoded by two EV*MDDs) from exponential (i.e., the size of the domain) to polynomial. For example, Figure 10 shows the pseudocode for an algorithm to perform the element-wise addition of two EV*MDDs, that is when $\alpha, \beta \geq 0$ and a, b are EV*MDD nodes at level L (unless $\alpha = 0$, in which case $a = \Omega$, or $\beta = 0$, in which case $b = \Omega$), $\textsc{Sum}(L, \langle\alpha,a\rangle, \langle\beta,b\rangle)$ returns an EV*MDD $\langle\rho,r\rangle$ such that, for all $x = (x_1, \ldots, x_L) \in X$, we have $\rho \cdot f_r(x) = \alpha \cdot f_a(x) + \beta \cdot f_b(x)$; of course, the input EV*MDDs are assumed to be in canonical form, and the output EV*MDD is guaranteed to be in the same canonical form. Its complexity is the product of the sizes of the input EV*MDDs.

In a practical implementation, the "unique table," which stores nodes and avoids duplicates, is implemented as a lossless hash table which, given a lookup key $\langle level, r[0], \ldots, r[n_k - 1]\rangle$, returns a node's address, while the cache is implemented as a (possibly) lossy hash table. The cache can be made more effective by scaling and exploiting commutativity. For example, by defining an arbitrary order on nodes (for example, $a \prec b$ if the memory address of a is smaller than that of b), we can exchange the two input EV*MDDs to ensure that $a \prec b$ prior to cache lookup, and then observe that $\alpha \cdot f_a + \beta \cdot f_b = \alpha \cdot (f_a + \gamma \cdot f_b)$, for $\gamma = \beta/\alpha$, so that we just store entries of the form $\langle SUM, a, \gamma, b \rightarrow \rho, r\rangle$ in the cache. Then, assuming $p \prec q$, the call $\textsc{Sum}(k, \langle 0.5,p\rangle, \langle 0.2,q\rangle)$ would be cached as $\langle SUM, p, 0.4, q \rightarrow \sigma, s, \rangle$ and

```
function Sum(level $k$, EV*MDD$\langle\alpha,a\rangle$, EV*MDD$\langle\beta,b\rangle$)
    if $a = b$ then return $\langle\alpha+\beta,a\rangle$   ▷ This includes the terminal case $k = 0$: $a = b = \Omega$
    if $\alpha = 0$ then return $\langle\beta,b\rangle$   ▷ $a = \Omega$ by definition
    if $\beta = 0$ then return $\langle\alpha,a\rangle$   ▷ $b = \Omega$ by definition
    if cache contains $\langle SUM,\alpha,a,\beta,b \to \rho,r\rangle$ then return $\langle\rho,r\rangle$   ▷ Check if result is in the cache
    $r \leftarrow$ NewNode($k$)   ▷ Create new temporary result node at level $k$
    for all $x_k \in X_k$ do
        $r[x_k] \leftarrow$ Sum($k-1$,$\langle\alpha \cdot a[x_k].val,a[x_k].ch\rangle$,$\langle\beta \cdot b[x_k].val,b[x_k].ch\rangle$)   ▷ Recurse down one level
    $\rho \leftarrow \max_{x_k \in X_k}\{r[x_k].val\}$;   ▷ Maximum edge value for node $r$ before normalization
    for all $x_k \in x_k$ do
        $r[x_k].val \leftarrow r[x_k].val/\rho$   ▷ Normalize node $r$ so that the maximum edge value is 1
    $r \leftarrow$ UniqueTableInsert($r$);   ▷ If node like $r$ exists, return it and delete $r$, else return $r$
    Enter $\langle SUM,\alpha,a,\beta,b \to \rho,r\rangle$ in cache;   ▷ Remember the result in the operation cache
    return $\langle\rho,r\rangle$;
```

Figure 10: Pseudo-code for sum of quasi-reduced EV*MDDs (a and b are either Ω or nodes at level k). The fully-reduced version is similar but slightly more involved, as it needs to take into account the levels of a and b.

return $\langle 0.5\sigma,s\rangle$, while a subsequent call Sum(k, $\langle 0.25,p\rangle$, $\langle 0.1,q\rangle$) or Sum(k, $\langle 0.1,q\rangle$, $\langle 0.25,p\rangle$) would find $\langle SUM, p, 0.4, q \to \sigma, s, \rangle$ in the cache and immediately return $\langle 0.25\sigma,s\rangle$.

4.5 Encoding Transition Rate Matrices with EV*MDDs

We now turn to the use of EV*MDDs to compactly encode the transition rate matrix $\boldsymbol{R}$ (the same as the intensity matrix $\boldsymbol{Q}$, but without the diagonal) of a CTMP. This can be accomplished using various approaches.

4.5.1 Monolithic Encoding vs. Disjunctive Partition Encoding

Clearly, a node r of a $2L$-level EV*MDD can encode an arbitrary function of the form $X \times X \to [0,1]$. Then, for $\sigma \geq 0$, the pair $\langle\sigma,r\rangle$ encodes an arbitrary function of the form $X \times X \to [0,\sigma]$, where EV*MDD levels $(1,\ldots,2L)$ correspond to state variables $(x'_1, x_1, \ldots, x'_L, x_L)$, that is, we use an interleaved order to describe the transition rate from x to x'. With a *monolithic* approach, we can then store $\boldsymbol{R}$ using a single EV*MDD $\langle\sigma,r\rangle$ where σ is the largest rate in $\boldsymbol{R}$ and r encodes matrix $\boldsymbol{R}/\sigma$.

However, many practical systems exhibit *asynchronous* behavior, that is each state change is due to some *event* $e \in E$ occurring (asynchronously) somewhere in the system. In these situations, we can employ a *disjunctive partition* to encode $\boldsymbol{R}$, storing a set of EV*MDDs $\{\langle\sigma_e,r_e\rangle : e \in E\}$, so that $\langle\sigma_e,r_e\rangle$ encodes matrix $\boldsymbol{R_e}$, where $R_e(x,x')$ describes the rate at which the system moves from state x to state x' due to the occurrence of event e.

With this disjunctive partition encoding, $\boldsymbol{R} = \sum_{e \in E} \boldsymbol{R_e}$ does not have to be built explicitly; rather, the individual matrices $\boldsymbol{R_e}$ are directly used in the numerical computations used to solve the CTMP. The idea of a disjunctive partition was initially suggested for BDDs (Burch, Clarke, & Long, 1991), although it is obviously also related to Kronecker encodings: consider Equation 76, which expresses $\boldsymbol{R}$ first and foremost as a sum.

The choice between a monolithic or a disjunctive partition encoding is largely model-dependent. In most applications, the high-level language description of the model suggests what the set of asynchronous events E should be. Thus, we first build the EV*MDDs for the disjuncts $\boldsymbol{R_e}$ then, if desired, we can explicitly build the EV*MDD for $\boldsymbol{R}$ by summing the EV*MDDs for the disjunct corresponding to each event (using the algorithm in Figure 10, for instance).

However, while the disjunct EV*MDDs are usually quite compact, the EV*MDD for $\boldsymbol{R}$ obtained by summing the disjuncts $\boldsymbol{R_e}$ might still be very compact, or it might grow very large. In the former case, the monolithic approach is preferable, as it allows us to directly use the EV*MDD encoding $\boldsymbol{R}$ in the numerical iterations. In the latter case, the disjunctive partition approach is preferable, as it allows us to use the EV*MDD for each $\boldsymbol{R_e}$ individually, without even attempting to build the monolithic EV*MDD encoding $\boldsymbol{R}$.

For example, consider a simple system of four Boolean variables, X_1, X_2, X_3, and X_4, and two events. The first event, c_{21}, changes the value of (X_2, X_1), interpreted as a 2-bit integer, in the sequence $0 \xrightarrow{[1]} 1 \xrightarrow{[1/2]} 2 \xrightarrow{[1/4]} 3 \xrightarrow{[1/8]} 0 \xrightarrow{[1]} \cdots$, where the numbers in the square brackets indicate the rate of the corresponding transition. The second event, c_{43}, changes the value of (X_4, X_3), interpreted as a 2-bit integer, in the sequence $0 \xrightarrow{[3]} 1 \xrightarrow{[1]} 2 \xrightarrow{[1/3]} 3 \xrightarrow{[1/9]} 0 \xrightarrow{[3]} \cdots$. Figure 11 on the left shows the EV*MDDs encoding the matrices $\boldsymbol{R_{21}}$ and $\boldsymbol{R_{43}}$ corresponding to these two events, as well as the matrix $\boldsymbol{R} = \boldsymbol{R_{21}} + \boldsymbol{R_{43}}$. (for visual simplicity, edges with value 0, which by definition point to the terminal node Ω, are not shown).

4.5.2 Adopting Ideas from Kronecker Encodings: Identity Patterns

Neither the monolithic approach nor the disjunctive partition approach exploit *locality*: the fact that most events (synchronously) affect only a few state variables. In other words, while each matrix $\boldsymbol{R_e}$ is conceptually of size $|X| \times |X|$, it usually has a much smaller *support* $S_e \subseteq \{x_1, \ldots, x_L\}$. Specifically, $X_k \in S_e$ if and only if X_k and e are *dependent*: the local state x_k affects the rate at which e occurs (including the case where it may disable e altogether, that is set its rate to 0) or is changed by the occurrence of e. When $X_k \notin S_e$, X_k and e are *independent* and the EV*MDD encoding $\boldsymbol{R_e}$ contains *identity patterns* in correspondence to x_k. For example, the EV*MDD encoding $\boldsymbol{R_{21}}$ in Figure 11 on the left exhibits such patterns with respect to variables X_4 and X_3, while the one for $\boldsymbol{R_{43}}$ exhibits them for X_2 and X_1.

Essentially, these identity patterns simply describe the fact that the value of x'_k (the new value of x_k after the occurrence of e) equals the old value of X_k and that the rate is not affected by the value of X_k, and this is true for all possible values of X_k. When this happens, the Kronecker encoding of event e has $\boldsymbol{R_e^{(k)}} = \boldsymbol{I_{n_k}}$, while the quasi-reduced or the fully-reduced forms alone cannot take advantage of these common patterns. To exploit these patterns, a combination of the fully-reduced form, for unprimed level X_k, and a new

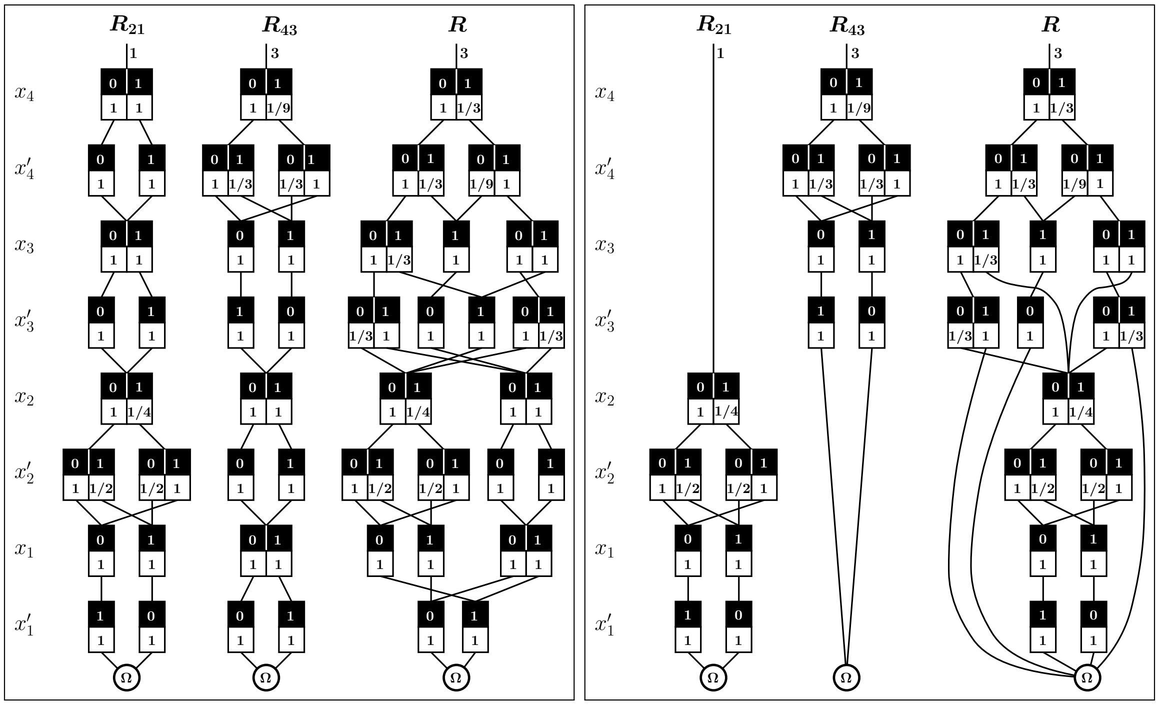


Figure 11: An example of EV*MDDs encoding transition rate matrices using the quasi-reduced form (left) or the fully-identity-reduced form (right). Omitted edges have implied value 0 (thus resulting in value 0 for any path containing them). The right diagrams are the same as those on the left, except that identity patterns have been omitted and are implied: Any completely skipped pair of levels is assumed to have an identity structure (compare to corresponding diagram on the left).

identity-reduced form (Ciardo & Yu, 2005) for primed level X'_k, is needed. This allows us to encode $\boldsymbol{R_e}$ in an EV*MDD which has nodes only at (unprimed and primed) levels corresponding to state variables in S_e. A further advantage of this *fully-identity-reduced* form is that the resulting decision diagrams, unlike a Kronecker encoding, also recognize and exploit *partial* identity patterns (those arising in models where X_k remains unchanged after e occurs in certain states but not in others). Figure 11 on the right shows the encoding of the same matrices $\boldsymbol{R_{21}}$, $\boldsymbol{R_{43}}$, and $\boldsymbol{R}$, but using this new fully-identity-reduced form.

4.5.3 Beyond Kronecker: Disjunctive-then-Conjunctive Partition Encoding

We can push the decomposition further by employing a *disjunctive-then-conjunctive partition* approach. This idea was first introduced for logic analysis (Ciardo & Yu, 2005) but it is also related to Equation 76, which expresses $\boldsymbol{R}$ as a sum of products. This is particularly appropriate for *globally-asynchronous locally-synchronous* systems, where not only each state change is due to an (asynchronous) event $e \in E$, but the occurrence of e depends on and (synchronously) changes only a few state variables. Each $\boldsymbol{R_e}$ is then further decomposed into the product of m matrices, $\boldsymbol{R_e} = \prod_{1\le c\le m} \boldsymbol{R_e^{(c)}}$ and, again, each matrix $\boldsymbol{R_e^{(c)}}$ is conceptually of size $|X| \times |X|$ but, in practice, it usually has a small support $S_e^{(c)}$. Specifically, $X_k \in S_e^{(c)}$ if and only if the fully-identity-reduced EV*MDD for $\boldsymbol{R_e^{(c)}}$ contains a node associated to X_k or X'_k. We restrict ourselves to the case where the supports of the conjuncts for an event are disjoint, so that $\bigcup_{1\le c\le m} S_e^{(c)} = S_e$ and each $S_e^{(c)}$ is substantially smaller than S_e. For example, when a Kronecker encoding for $\boldsymbol{R_e}$ exists, that is $\boldsymbol{R_e} = \bigotimes_{1\le k\le L} \boldsymbol{R_e^{(k)}}$, we have $S_e^{(k)} = \{X_k\}$ for each $X_k \in S_e$, that is for each $\boldsymbol{R_e^{(k)}} \neq \boldsymbol{I_{n_k}}$; in this case, a disjunctive-then-conjunctive approach that uses an EV*MDD to store each $\boldsymbol{R_e^{(c)}}$ is as compact as the disjunctive partition approach that uses an EV*MDD to store each $\boldsymbol{R_e}$, and both are essentially just as compact as the Kronecker approach (except that they can save additional memory by exploiting partial identity patterns).

The disjunctive-then-conjunctive approach is instead distinctly more efficient when the Kronecker approach is not applicable (that is, when the Kronecker approach would require an enormous set of events to correctly describe $\boldsymbol{R}$), but it can nevertheless be seen as an extension of the Kronecker approach. Consider using the decomposition of Equation 75:

$$\begin{aligned}\boldsymbol{R_e} = \bigotimes_{1\le k\le L} \boldsymbol{R_e^{(k)}} &= \prod_{1\le k\le L} \boldsymbol{P_{n_{1:k},n_{k+1:L}}}^{\top} \cdot (\boldsymbol{I_{\overline{n}_k}} \otimes \boldsymbol{R_e^{(k)}}) \cdot \boldsymbol{P_{n_{1:k},n_{k+1:L}}} \\ &= \prod_{X_k\in S_e} \boldsymbol{P_{n_{1:k},n_{k+1:L}}}^{\top} \cdot (\boldsymbol{I_{\overline{n}_k}} \otimes \boldsymbol{R_e^{(k)}}) \cdot \boldsymbol{P_{n_{1:k},n_{k+1:L}}}\end{aligned}$$

where the last step simply stresses that, when $\boldsymbol{R_e^{(k)}} = \boldsymbol{I_{n_k}}$, the corresponding factor is just $\boldsymbol{I_{n_{1:L}}}$ and can be skipped.

This is the idea behind the *Shuffle Algorithm* (Fernandes, Plateau, & Stewart, 1998), which, as observed by Buchholz, Ciardo, Donatelli, and Kemper (2000), is very efficient, but only when the matrices $\boldsymbol{R_e^{(k)}}$ are not "too sparse." (The perfect shuffle pre- and post-multiplications are essentially free; they simply describe a different state indexing.)

Then, the disjunctive-then-conjunctive approach extends the Kronecker expression of $\boldsymbol{R_e}$ to allow situations where the factors are not restricted to a support consisting of just one variable, but still exploits each factor's locality:

$$\boldsymbol{R_e} = \prod_{1\le c\le m} \boldsymbol{P_{S_e^{(c)}}}^{\top} \cdot \left(\boldsymbol{I_{\overline{S}_e^{(c)}}} \otimes \boldsymbol{R_{S_e^{(c)}}}\right) \cdot \boldsymbol{P_{S_e^{(c)}}} \qquad (77)$$

where $\boldsymbol{P_{S_e^{(c)}}}^{\top}$ and $\boldsymbol{P_{S_e^{(c)}}}$ are perfect shuffle permutations that respectively move all dependent state variables in $S_e^{(c)}$ at the end of the variable order and back to their original position, $\boldsymbol{I_{\overline{S}_e^{(c)}}}$ is the identity matrix of size $\prod_{X_h\notin S_e^{(c)}} n_h$ (the skipped levels), and $\boldsymbol{R_{S_e^{(c)}}}$ is a square matrix of size $\prod_{X_h\in S_e^{(c)}} n_h$ (the conjunct encoded by an EV*MDD if we ignore the skipped levels corresponding to state variables not in $S_e^{(c)}$).

Since the supports $S_e^{(c)}$ are disjoint, this generalization of the Kronecker approach comes at no additional cost and essentially corresponds to a Kronecker approach where we allow each event to be defined on a different set of state variables, each set corresponding to a different partition of the "basic" state variables $(X_1, .., X_L)$. In this case, building the EV*MDD for $\boldsymbol{R_e}$ by multiplying the EV*MDDs for each $\boldsymbol{R}_{\boldsymbol{S_e^{(c)}}}$ does not involve any numeric multiplication, but it grows the size of the diagram if the spans of the sets $S_e^{(c)}$ are not disjoint; for example, if $S_e^{(1)} = \{X_3, X_7\}$ and $S_e^{(2)} = \{X_4, X_6\}$, then each path from X_7 to X_3 in the EV*MDD for $\boldsymbol{R_e}$ will contain a copy of the entire EV*MDD encoding $\boldsymbol{R}_{\boldsymbol{e^{(2)}}}$.

4.5.4 Numerical Solutions with Decision Diagrams

We have described a method for storing the rate or intensity matrix compactly for common process models. We now address the use of these data structures in CTMP computations. The literature surrounding decision diagrams and CTMPs is primarily concerned with computation of the unconditional distribution of the resulting process, either at a finite time or (more commonly) in the limit of infinite time (the stationary distribution). We follow the convention of this literature and refer to this as the *solution* of the process. Model estimation, solutions conditioned on evidence, and computations of marginals or other statistics are, to our knowledge, unexplored for these representations, but we return to this later.

When matrix $\boldsymbol{R}$ is stored using $2L$-level EV*MDDs (using a monolithic, a disjunctive partition, or a disjunctive-then-conjunctive partition encoding), the traditional numerical solution algorithms need to be adjusted accordingly. First of all, when seeking an *exact* solution, neither the stationary vector $\boldsymbol{\pi}$ nor the transient vector $\boldsymbol{\pi}(t)$ admit a compact EV*MDD representation (unless the modeled system contains extensive symmetries or is composed of completely independent subsystems, which is rarely the case in practice).

Two approaches have been explored. For an exact solution for the stationary distribution $\boldsymbol{\pi}$ (the null-space of $\boldsymbol{Q}$), a *hybrid* approach (Kwiatkowska, Norman, & Parker, 2004) is usually best, where the solution is stored as a full vector of reals having size equal to the number of reachable states ($|X_{rch}|$, equal to $|X|$ only if all states are reachable) while the rate matrix $\boldsymbol{R}$ is stored with EV*MDDs, and the *expected holding time* vector h (the inverse absolute values of the diagonal of $\boldsymbol{Q}$) is stored either as a full vector or with EV*MDDs. Clearly, such an approach scales the size of the tractable problems by eliminating the main memory obstacle (the storage of $\boldsymbol{R}$), only to encounter the next memory obstacle (the storage of the solution vector). For example, Figure 12 shows the pseudocode for a classic Jacobi-style stationary solution of an ergodic CTMP when the transition rate matrix is monolithically encoded by the EV*MDD $\langle\sigma, r\rangle$, while the state space X_{rch} is indexed by an EV$^+$MDD $\langle 0, p\rangle$, as previously discussed, so that, for each $i \in X$, we can compute $\phi_{X_{rch}}(i)$, an index between 0 and $|X_{rch}| - 1$ included if $i \in X_{rch}$, or ∞ if $i \notin X_{rch}$. Function $\phi_{X_{rch}}$ is used to index entries of the solution vector: $\boldsymbol{\pi^{new}}$ and $\boldsymbol{\pi^{old}}$. The holding time vector is stored as the full vector h, also indexed by $\phi_{X_{rch}}$ (but it could have been stored using EV*MDDs instead). At each recursive call of JacobiRecur, we descend a "from" and a "to" level from the current rate matrix EV*MDD node and a single level from the corresponding "source" and "destination" EV$^+$MDD nodes (these are needed to index the full vectors of reals, and are initially both set to $\langle 0, p\rangle$, encoding the entire $\phi_{X_{rch}}$ function). Note that, in the simple case when all states are reachable (that is, $X = X_{rch}$) the indexing function $\phi_{X_{rch}}$ is just the mixed-base value $\phi(i) = \sum_{1 \le k \le L} i_k \cdot n_{1:k-1}$ discussed in Section 4.3 and, as such, it does not really require an EV$^+$MDD for its encoding; on the other hand, as shown in Figure 9, this EV$^+$MDD is just a single path of nodes, so its use does not carry

```
▷ Computes $\boldsymbol{\pi}$ such that $\boldsymbol{\pi Q} = \mathbf{0}$. $\langle\sigma,r\rangle$ is $\boldsymbol{R}$, $\langle 0,p\rangle$ is $\phi_{X_{rch}}$
function JacobiIteration(EV*MDD$\langle\sigma,r\rangle$,EV$^+$MDD$\langle 0,p\rangle$)
    $\boldsymbol{\pi^{old}} \leftarrow$ "initial guess"        ▷ Real vector of size $|X_{rch}|$, visible to JacobiRecur
    $num\_iter \leftarrow 0$
    repeat
        $\boldsymbol{\pi^{new}} \leftarrow$ zero vector        ▷ Real vector of size $|X_{rch}|$, visible to JacobiRecur
        JacobiRecur($L$,$\langle\sigma,r\rangle$,$\langle 0,p\rangle$,$\langle 0,p\rangle$)
        for all $i \in \{0,\ldots,|X_{rch}|-1\}$ do
            $\boldsymbol{\pi^{new}}[i] \leftarrow \boldsymbol{\pi^{new}}[i] \cdot \boldsymbol{h}[i]$        ▷ $\boldsymbol{h}$ is the holding time vector
        swap($\pi^{old}$, $\pi^{new}$)
        $num\_iter \leftarrow num\_iter + 1$
    until $num\_iter > MAX\_ITER$ or Converged($\boldsymbol{\pi^{old}}$,$\boldsymbol{\pi^{new}}$)
    ▷ for example, using a relative or absolute test

▷ Computes $\boldsymbol{\pi^{new}} \leftarrow \boldsymbol{\pi^{old}R}$
function JacobiRecur(level $k$, EV*MDD$\langle\sigma,m\rangle$, EV$^+$MDD$\langle\eta_{src},src\rangle$,
EV$^+$MDD$\langle\eta_{des},des\rangle$)
    if $src = des = \Omega$ then
        $\boldsymbol{\pi^{new}}[\eta_{des}] \leftarrow \boldsymbol{\pi^{new}}[\eta_{des}] + \boldsymbol{\pi^{old}}[\eta_{src}] \cdot \sigma$
        return
    for $i$ from 0 to $n_k-1$ s.t. $m[i].val \neq 0$ and $src[i].val \neq \infty$ do        ▷ "from" level $k$
        for $j$ from 0 to $n_k-1$ s.t. $m[i][j].val \neq 0$ and $des[j].val \neq \infty$ do    ▷ "to" level $k'$
            $\eta'_{src} \leftarrow \eta_{src} + src[i].val$
            $\eta'_{des} \leftarrow \eta_{des} + des[j].val$
            $\sigma' \leftarrow \sigma \cdot m[i][j].val$
            JacobiRecur($k-1$, $\langle\sigma',m[i][j].ch\rangle$, $\langle\eta'_{src},src[i].ch\rangle$, $\langle\eta'_{des},des[j].ch\rangle$)
```

Figure 12: A Jacobi-style iteration for the stationary solution ($\boldsymbol{\pi Q} = \frac{d\boldsymbol{\pi}}{dt} = \mathbf{0}$) when $\boldsymbol{R}$ (non-diagonal elements of $\boldsymbol{Q}$) is stored as a monolithic EV*MDD and $\boldsymbol{h}$ (inverse absolute value of the diagonal elements of $\boldsymbol{Q}$) is stored in a vector. $\langle\sigma,r\rangle$ is the encoding of $\boldsymbol{R}$ and $\langle 0,p\rangle$ is the encoding of the mapping from states to indices (for $\boldsymbol{\pi}$ and $\boldsymbol{h}$).

any overhead. A similar hybrid approach can be used to compute a transient solution using a uniformization-style algorithm where $\boldsymbol{R}$ is also stored using EV*MDDs but, again, the size of the full vectors limits scalability.

The filtering and smoothing operations as described in Section 2.5 have not been explicitly tackled for decision-diagram encodings. However, if we are willing to represent the distribution exactly (as above), we can do the necessary vector-matrix multiplications directly on the decision-diagrams (without expanding them). Estimating an EV*MDD representation of $\boldsymbol{R}$ from data is completely unexplored.

To tackle larger problems we must instead be willing to accept an approximate solution. However, work in this area is mostly restricted to systems exhibiting special structures.

One exception is the work of Wan et al. (2011), which addresses the stationary solution of arbitrary ergodic CTMPs whose state space is encoded as an MDD and whose transition rate matrix is encoded as one or more EV*MDDs. The approach uses L different approximate aggregations of the exact CTMP and solves them iteratively, until reaching a fixpoint. This approach provides an exact solution under certain conditions — essentially, if the system has a so-called "product-form" (Baskett, Chandy, Muntz, & Palacios-Gomez, 1975). Unfortunately, no similar approximation for the transient solution of a structured CTMP has been proposed so far.

Thus for state spaces large enough that a single vector over their values cannot be maintained, the literature for inference and estimation with these models is very limited. However, in the next section we describe a different model that can be viewed as a restricted form of the disjunctive EV*MDD encoding of this section. This model has many inference and estimation method and we believe this link between the two may allow for those methods to be extended to more general decision-diagram representations.

5. Continuous-Time Bayesian Networks

In the artificial intelligence and machine learning literatures, continuous-time Bayesian networks (CTBNs) (Nodelman, Shelton, & Koller, 2002) were developed as an extension of dynamic Bayesian networks (DBNs). As we discuss in this section, they are a more limited case of the disjunctive EV*MDD encodings above. However, approximate methods and computations conditioned on evidence have been more extensively developed for CTBNs.

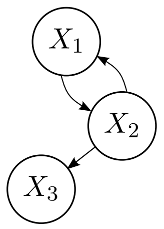
A CTBN consists of a set of variables, $\{X_1, X_2, \ldots, X_L\}$, a directed graph $\mathcal{G}$ with a one-to-one mapping between the nodes and the variables, a set of conditional intensity matrices for each variable, and an initial distribution. The initial distribution is usually described as a Bayesian network (to keep its description compact), but many of the algorithms and theory hold for other compact distribution representations.

The graph $\mathcal{G}$ describes instantaneous influence of variables on each other. An edge from X_i to X_j denotes that the rates of transitions of X_j depend on the instantaneous value of X_i. Note that $\mathcal{G}$ may be cyclic.

These dependent rates are captured in the conditional intensity matrices. Let n_i be the number of states for variable X_i. We denote the parents of variable X_i as Par_i and a joint assignment to Par_i as $\mathbf{par}_{\boldsymbol{i}}$. The set of conditional intensity matrices for variable X_i consists of one n_i-by-n_i intensity matrix for each possible instantiation $\mathbf{par}_{\boldsymbol{i}}$: $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}$ for which we denote element x_i, x_i' as $q_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}$, the rate of X_i transitioning from x_i to x_i' when Par_i have values $\mathbf{par}_{\boldsymbol{i}}$.

Semantically, a CTBN is a continuous-time Markov process over the joint state space of all constituent variables. We let $\delta(\boldsymbol{x}, \boldsymbol{x'})$ be equal to the set of variables whose assignments differ between joint assignments $\boldsymbol{x}$ and $\boldsymbol{x'}$. The joint intensity matrix for the entire process can be described as

$$q_{\boldsymbol{x},\boldsymbol{x'}} = \begin{cases} \sum_{i=1}^{L} q_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}} & \text{if } \delta(\boldsymbol{x}, \boldsymbol{x'}) = \{\} \\ q_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}} & \text{if } \delta(\boldsymbol{x}, \boldsymbol{x'}) = \{X_i\} \\ 0 & \text{otherwise} \end{cases} \tag{78}$$



$$\boldsymbol{Q}_{\boldsymbol{X_1}|\boldsymbol{X_2=0}} = \begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix}$$

$$\boldsymbol{Q}_{\boldsymbol{X_1}|\boldsymbol{X_2=1}} = \begin{bmatrix} -4 & 4 \\ 3 & -3 \end{bmatrix} \quad \boldsymbol{Q}_{\boldsymbol{X_3}|\boldsymbol{X_2=0}} = \begin{bmatrix} -9 & 0 & 9 \\ 8 & -10 & 2 \\ 7 & 4 & -11 \end{bmatrix}$$

$$\boldsymbol{Q}_{\boldsymbol{X_2}|\boldsymbol{X_1=0}} = \begin{bmatrix} -5 & 5 \\ 6 & -6 \end{bmatrix} \quad \boldsymbol{Q}_{\boldsymbol{X_3}|\boldsymbol{X_2=1}} = \begin{bmatrix} -12 & 6 & 6 \\ 8 & -13 & 5 \\ 7 & 7 & -14 \end{bmatrix}$$

$$\boldsymbol{Q}_{\boldsymbol{X_2}|\boldsymbol{X_1=1}} = \begin{bmatrix} -7 & 7 \\ 8 & -8 \end{bmatrix}$$

Figure 13: An example CTBN graph. See Figures 14, 15, and 17 for the same CTMP in other representations.

where $\mathbf{par}_i$ is the assignment to Par_i in $\boldsymbol{x}$. The intensity of transition between two states that differ only by one variable can be read from the appropriate conditional intensity matrix for that variable. The intensity of transition between any other two states (that differ by more than one variable) is zero. The diagonal elements are filled in to be the negative row sums. This process allows two variables to transition at arbitrarily close times, but not at exactly the same time.

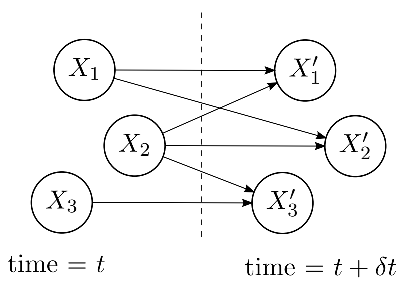
A CTBN retains the local Markov properties of a standard Bayesian network. In particular, a variable (local process) is independent of its non-descendants, given its parents. Of course, because of cycles, parents may also be descendants, but this does not pose a problem to the definition. Note, however, that *given* refers to conditioning on the entire trajectory of a variable (from the starting time until the ending time, after which no variables are queried or observed). Conditioning on the current value is not sufficient (even for rendering only the current values independent).

The global Markov properties also hold. The Markov blanket for a variable is the union of the sets of its parents, its children, and its children's parents. Note that these sets can have significant overlap, as cycles are permitted. Conditioned on its Markov blanket, a variable is independent of all other variables.

5.1 Connections to Other Representations

A CTBN can be related to a number of other representations. For instance, Portinale and Codetta-Raiteri (2009) link CTBNs to stochastic Petri nets (Ajmone Marsan, Balbo, Conte, Donatelli, & Franceschinis, 1995). Donatelli (1994) shows the translation from stochastic Petri nets to Kronecker operators and Ciardo, Zhao, and Jin (2012) show the translation from (ordinary, timed, or stochastic) Petri nets to various classes of decision diagrams. However, below we concentrate on more direct comparisons of the approaches presented in this tutorial.

Figure 13 shows a simple small CTBN of two binary variables (X_1 and X_2) and one ternary variable (X_3). We will use this as a running example for how to convert from a CTBN to other compact representations.



$P(X_1'|X_1, X_2)$:

	0,0	1,0	0,1	1,1
0	$1-1\delta t$	$2\delta t$	$1-4\delta t$	$3\delta t$
1	$1\delta t$	$1-2\delta t$	$4\delta t$	$1-3\delta t$

$P(X_2'|X_2, X_1)$:

	0,0	1,0	0,1	1,1
0	$1-5\delta t$	$6\delta t$	$1-7\delta t$	$8\delta t$
1	$5\delta t$	$1-6\delta t$	$7\delta t$	$1-8\delta t$

$P(X_3'|X_3, X_2)$:

	0,0	1,0	2,0	0,1	1,1	2,1
0	$1-9\delta t$	$8\delta t$	$7\delta t$	$1-12\delta t$	$8\delta t$	$7\delta t$
1	0	$1-10\delta t$	$4\delta t$	$6\delta t$	$1-13\delta t$	$7\delta t$
2	$9\delta t$	$2\delta t$	$1-11\delta t$	$6\delta t$	$5\delta t$	$1-14\delta t$

Figure 14: A DBN whose limit as $\delta t \to 0$ approaches the CTBN of Figure 13.

5.1.1 Connection to DBN

From a CTBN, we can construct a dynamic Bayesian network (DBN) whose parameters are a function of the time between slices such that its limit as this time-slice width approaches zero is the original CTBN. In particular, the DBN has no intra-time-slice edges (this is because two variables in the CTBN cannot change at exactly the same time). If X_i has parents Par_i in the CTBN, then it has the same parents (at the previous time slice) plus its previous value in the DBN. Figure 14 shows the CTBN of Figure 13 as a DBN. If X_i in the CTBN has an intensity matrix $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_i}$ for parent values $\mathbf{par}_i$, then the corresponding variable in the DBN has the conditional probability distribution

$$p_{\mathrm{DBN}}(x_i'|x_i, \mathbf{par}_i) = \delta_{x_i', x_i} + \delta t\, q_{x_i, x_i'|\mathbf{par}_i} \tag{79}$$

where x_i' is the value of X_i at the “next” time step (and x_i and $\mathbf{par}_i$ are the values at the “previous” time step), δ_{x_i', x_i} is 1 if $x_i' = x_i$ and 0 otherwise, and δt is the time between time slices. The limit of this process as δt approaches 0 is the original CTBN.

5.1.2 Connection to Kronecker Algebra

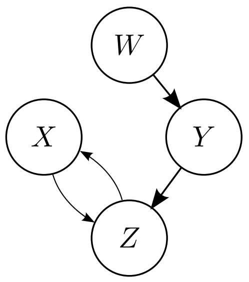
The joint intensity matrix expressed in Equation 78 can also be written as a sum of Kronecker products. We need first to define a conceptually simple, but notationally cumbersome, term. First, let $\boldsymbol{\Delta_{i,j}}$ be a matrix of all 0, except for a single 1 in location i, j (same as before). Second, let $\tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}|\mathbf{par}_i}$ denote the Kronecker product of one matrix for each variable in the CTBN. If the variable is X_i, the matrix is $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_i}$. If the variable is a parent of X_i and has value x_k in $\mathbf{par}_i$, then the matrix is $\boldsymbol{\Delta}_{\boldsymbol{x_k}, \boldsymbol{x_k}}$. Otherwise, the matrix is the identity matrix. In this way this Kronecker product distributes the elements of $\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_i}$ to the relevant entries of the joint intensity matrix.

We can now define the joint intensity matrix. Let $\tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}}$ be $\sum_{\mathbf{par}_i} \tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}|\mathbf{par}_i}$. Then, $\boldsymbol{Q} = \sum_i \tilde{\boldsymbol{Q}}_{\boldsymbol{X_i}}$. Figure 15 gives an example for the CTBN of Figure 13. Figure 16 gives another example. While the Kronecker product in general does not handle the diagonal elements, the expansion works for the intensity matrix in this case, since only one of the matrices in each product is non-diagonal.

$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X_1}} = \sum_{x_2} \boldsymbol{Q}_{\boldsymbol{X_1|x_2}} \otimes \boldsymbol{\Delta}_{\boldsymbol{x_2,x_2}} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X_2}} = \sum_{x_1} \boldsymbol{\Delta}_{\boldsymbol{x_1,x_1}} \otimes \boldsymbol{Q}_{\boldsymbol{X_2|x_1}} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X_3}} = \sum_{x_2} \boldsymbol{I} \otimes \boldsymbol{\Delta}_{\boldsymbol{x_2,x_2}} \otimes \boldsymbol{Q}_{\boldsymbol{X_3|x_2}}$$

$$\boldsymbol{Q} = \tilde{\boldsymbol{Q}}_{\boldsymbol{X_1}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{X_2}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{X_3}}$$

Figure 15: Sum of Kronecker encoding of the rate matrix Q of the CTBN in Figure 13.



$$\tilde{\boldsymbol{Q}}_{\boldsymbol{W}} = \boldsymbol{Q}_{\boldsymbol{W}} \otimes \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{X}} = \sum_{z} \boldsymbol{I} \otimes \boldsymbol{Q}_{\boldsymbol{X|z}} \otimes \boldsymbol{I} \otimes \boldsymbol{\Delta}_{\boldsymbol{z,z}}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{Y}} = \sum_{w} \boldsymbol{\Delta}_{\boldsymbol{w,w}} \otimes \boldsymbol{I} \otimes \boldsymbol{Q}_{\boldsymbol{Y|w}} \otimes \boldsymbol{I}$$
$$\tilde{\boldsymbol{Q}}_{\boldsymbol{Z}} = \sum_{x,y} \boldsymbol{I} \otimes \boldsymbol{\Delta}_{\boldsymbol{x,x}} \otimes \boldsymbol{\Delta}_{\boldsymbol{y,y}} \otimes \boldsymbol{Q}_{\boldsymbol{Z|x,y}}$$

$$\boldsymbol{Q} = \tilde{\boldsymbol{Q}}_{\boldsymbol{W}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{X}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{Y}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{Z}}$$

Figure 16: Sum of Kronecker product encoding of a CTBN with more than one parent per node.

5.1.3 Connection to Decision Diagrams

The above decomposition of a CTBN into a sum of Kronecker products helps clarify the connection to the edge-valued decision diagrams of the previous section. A CTBN is a particularly structured version of the disjunctive EV*MDD encoding of Section 4.5.1 paired with the identity encoding of Section 4.5.2. In particular, a CTBN describes a CTMP which can also be described by a sum of EV*MDDs in fully-identity-reduced form. The two descriptions have the same order space complexity. The decision-diagram encoding has one EV*MDD for each variable and each joint value for its parents.

Figure 17 shows the disjunction of EV*MDDs for the CTBN in Figure 13. Each EV*MDD only encodes the non-identity matrices in the Kronecker product expression; the identity matrices are implied by the fully-identity-reduced form. As another example, for the CTBN in Figure 16, we could construct 9 EV*MDDs: 1 for W, 2 for each of X and Y, and 4 for Z.

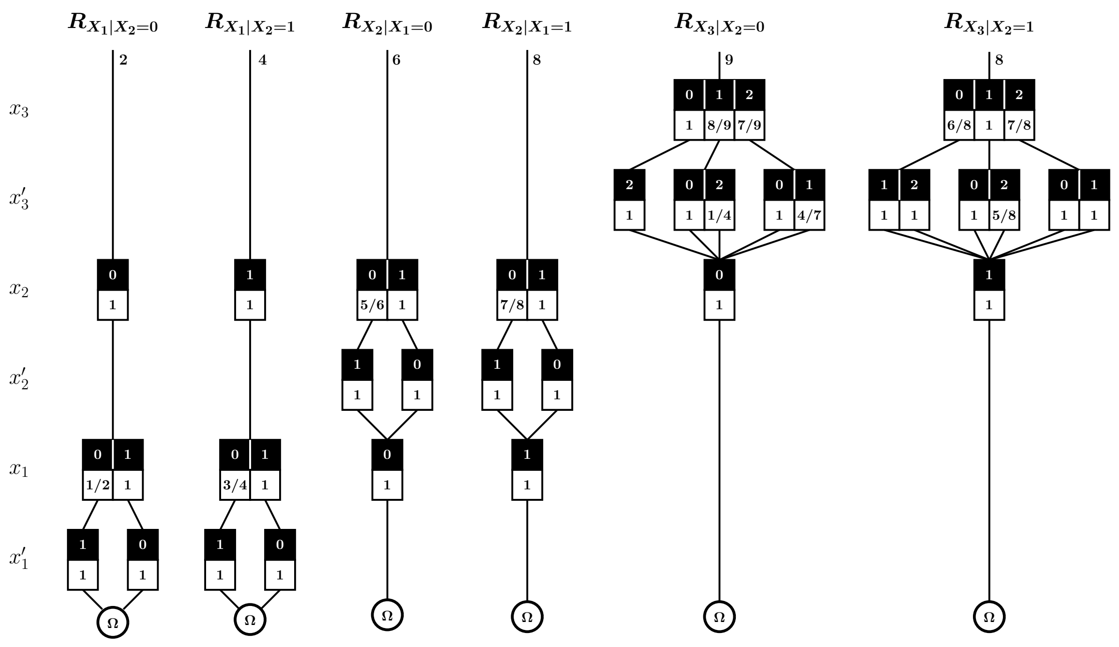


Figure 17: A set of identity-reduced EV*MDDs whose sum is the same as the CTBN of Figure 13.

Note that a disjunction of EV*MDDs can compactly encode structure *within* a variable's local rate matrix, while a CTBN cannot. In this way, they represent a generalization that can exploit context-sensitive independence.

Whether merging EV*MDDs for a given variable or merging EV*MDDs for multiple variables will result in a reduction or increase in the representation size is largely an empirical question. However, we would generally expect an increase in size because only one transition is allowed at a time, so the paths through the levels must remember whether any previous variable has changed and, if so, which one (for example, as in Figure 11).

5.2 Sampling

Sampling from a CTBN can be done by straight-forward application of the sampling method described in Section 2.1. We need not construct the full intensity matrix. Instead, for any joint assignment $\boldsymbol{x}$, we can find the diagonal element by summing the diagonals of the relevant conditional intensity matrices. This gives us the rate of the exponential to sample for the time to the next variable change. We can read the intensities for each variable's potential transitions from the relevant row of its conditional intensity matrix and we select a variable and the new state for that variable in proportion to the intensity. This process takes $O(L)$ time for each transition (where L is the number of variables).

We can do better by exploiting the racing and memoryless properties of exponential distributions (discussed in Section 2.1). To select which variable transitions next, we will race exponential distributions for each variable with rates of the corresponding diagonal

```
function SampleCTBN(CTBN, initial distribution $\pi_0$, end time $T$)
    Let $\mathcal{T}r \leftarrow$ empty trajectory
    Let $(x_1, x_2, \ldots, x_L) \leftarrow$ joint sample from $\pi_0$    ▷ As per algorithm for $\pi_0$'s representation
    for $i$ from 1 to $L$ do
        Add $(X_i = x_i$ @ $0)$ to $\mathcal{T}r$
    Let $t \leftarrow 0$
    Let $E \leftarrow$ an empty event priority queue of time-variable pairs.
    repeat
        for all variables $X_i$ that do not have an event in $E$ do
            Sample $\Delta t$ from an exponential with rate $q_{x_i|\mathbf{par}_i}$
            Add $\langle t + \Delta t, X_i \rangle$ to $E$
        Let $\langle t, X_i \rangle \leftarrow$ the earliest event in $E$    ▷ Update $t$ and get new variable to change
        if $t < T$ then
            Sample $x_i'$ from a multinomial proportional to $q_{x_i,x_i'|\mathbf{par}_i}$
            Let $x_i \leftarrow x_i'$    ▷ Update local copy of variable assignments
            Add $(X_i = x_i$ @ $t)$ to $\mathcal{T}r$
            Remove $X_i$ and all children of $X_i$ from $E$    ▷ Their times must be resampled
    until $t \geq T$
```

Figure 18: Algorithm to sample from CTBN

elements of their conditional intensity matrices. We then note that if a variable is not chosen, we can treat its transition time as two separate random draws: a draw stating that its transition time is after the chosen time and a draw stating when after the chosen transition time it will next transition (because of the memoryless property of the exponential distribution). That means that if the chosen transition did not affect the rate for the variable in question, we do not need to resample its transition time. By using a priority queue for transitions times (not durations), we can reduce the running time per transition to $O(D \ln L)$ where D is the maximal out-degree of the graph. This method is made explicit in Figure 18.

5.3 Inference

Inference in a CTBN is the process of calculating an expected value of the full trajectory, given some partial trajectory. The most basic case is to infer the conditional probability of a single variable at a single time point (the expectation of an indicator function) given a partial trajectory. There are many ways in which a trajectory may be partial. The most obvious for a variable-based model like a CTBN is to have variables only observed at particular times and intervals. Each variable can have its own observation times and intervals. Thus, for each variable, we assume we have evidence like that of Section 2.5.1: There are time points at which the variable has known values and there are time intervals during which the variable has known values (which might include observations of transitions).

Unfortunately, even if there is no evidence, this problem is NP-hard. In particular, deciding whether the marginal probability of a single value of a single variable at a single time point is greater than any positive threshold is NP-hard. This has been generally accepted, although never formally demonstrated. We provide the proof in the Appendix.

Thus, all known algorithms for CTBN inference have exponential (in the number of variables) running time. The simplest method is to treat the CTBN as a general CTMP with a single intensity matrix $\boldsymbol{Q}$. We can apply the forward and backward passes of Section 2.5. While the intensity matrix can be stored in compact form, the resulting vectors require space for each instantiation to every variable in the CTBN (exponential space). We need only keep values for states consistent with the evidence. Thus, if at all times only a few variables are unobserved, the inference is tractable. But, if there are periods during which many variables are unobserved, we require approximate inference methods (overviewed in Section 5.5).

Other than calculating the probability of a variable at a time, the other common case of inference is to calculate the expected sufficient statistics. As shown in Section 5.4.1, this means calculating $\bar{N}[x_i, x'_i|\mathbf{par}_i]$ and $\bar{T}[x_i|\mathbf{par}_i]$ for all values of i, x_i, x'_i, and $\mathbf{par}_i$. The former is the expected number of times variable X_i transitioned from x_i to x'_i while its parents were in state $\mathbf{par}_i$, and the latter is the expected amount of time variable X_i was in state x_i while its parents were in state $\mathbf{par}_i$.

The proof for marginal calculation can easily be adapted to show that deciding whether these quantities are non-zero is also NP-hard. Therefore, the only known method is to again treat the system as a general CTMP with a single large Q matrix. We can then apply Equation 65 and Equation 64 to find the expected number of transitions and expected amount of time for any joint assignments. If we let $J(x_i, \mathbf{par}_i)$ be the set of joint assignments to all variables that are consistent with $X_i = x_i$ and $\mathrm{Par}_i = \mathbf{par}_i$, we can find the expected sufficient statistics for the CTBN as

$$\bar{T}[x_i|\mathbf{par}_i] = \sum_{\boldsymbol{x}\in J(x_i,\mathbf{par}_i)} \bar{T}[\boldsymbol{x}] \tag{80}$$

$$\bar{N}[x_i, x'_i|\mathbf{par}_i] = \sum_{\boldsymbol{x}\in J(x_i,\mathbf{par}_i)} \sum_{\boldsymbol{x'}\in J(x'_i,\mathbf{par}_i)} \bar{N}[\boldsymbol{x}, \boldsymbol{x'}] \tag{81}$$

5.4 Parameter and Graph Estimation

The initial distribution of a CTBN can be estimated separately using any standard method for estimation of a Bayesian network (or whatever other compact representation is desired). This requires only data about the value of the trajectory's value (or trajectories' values) at time 0.

We will concentrate on estimation of the rate parameters and dynamics graph structure ($\mathcal{G}$). This exposition will assume there is a single trajectory, $\mathcal{Tr}$. However, multiple trajectories can be used by summing their sufficient statistics.

5.4.1 Parameter Estimation

The set of CTBNs with a fixed graph structure is just a subset of the exponential family of CTMPs in which most parameters are fixed to 0 and many of the remaining ones are tied to each other (share the same value). Thus, the log-likelihood of Equation 30 applies here too, but where the sufficient statistics for tied parameters are summed:

$$\ln p_{\text{CTBN}}(\mathcal{T}r) = \ln P_0(\mathcal{T}r(0)) + \sum_{i,\mathbf{par}_{\boldsymbol{i}},x_i} \left(-T[x_i|\mathbf{par}_{\boldsymbol{i}}] q_{x_i|\mathbf{par}_{\boldsymbol{i}}} + \sum_{x'_i \neq x_i} N[x_i, x'_i|\mathbf{par}_{\boldsymbol{i}}] \ln q_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}} \right) \tag{82}$$

$$= \ln P_0(\mathcal{T}r(0)) + \sum_{i,\mathbf{par}_{\boldsymbol{i}},x_i,x'_i \neq x_i} \left(-T[x_i|\mathbf{par}_{\boldsymbol{i}}] q_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}} + N[x_i, x'_i|\mathbf{par}_{\boldsymbol{i}}] \ln q_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}} \right) \tag{83}$$

$$= \ln P_0(\mathcal{T}r(0)) + \sum_i l_{X_i}(\mathcal{T}r) \tag{84}$$

where i ranges over variables, $\mathbf{par}_{\boldsymbol{i}}$ ranges over joint assignments to the parents of i, and x_i and x'_i range over differing assignments to X_i. $T[x_i|\mathbf{par}_{\boldsymbol{i}}]$ denotes the amount of time $X_i = x_i$ while $\text{Par}_i = \mathbf{par}_{\boldsymbol{i}}$. Similarly, $N[x_i, x'_i|\mathbf{par}_{\boldsymbol{i}}]$ denotes the number of transitions of X_i from x_i to x'_i while $\text{Par}_i = \mathbf{par}_{\boldsymbol{i}}$. These new sufficient statistics are sums of the sufficient statistics of the flat CTMP, summing over all assignments to all CTBN variables in which X_i and Par_i remain the same (see Equations 80 and 81). Given a complete trajectory, we can construct them directly without employing such (exponentially large) sums. The last line above is by definition of l_{X_i}, the "local log-likelihood" of variable X_i. Note that this is a function only of the trajectories of X_i and its parents (not of all of $\mathcal{T}r$).

Maximizing Equation 83 is a straight-forward extension of maximizing Equation 30:

$$\hat{q}_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}} = N[x_i, x'_i|\mathbf{par}_{\boldsymbol{i}}]/T[x_i|\mathbf{par}_{\boldsymbol{i}}] \ . \tag{85}$$

We can produce Bayesian posterior distributions over the parameters if we take independent conjugate prior distributions over each $q_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}}$ parameter (Nodelman et al., 2003). Just as for a flat CTMP, our conjugate prior is a gamma distribution with hyper-parameters $\alpha_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}}$ and $\tau_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}}$ for parameter $q_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}}$. The resulting posterior is also a gamma distribution with corresponding hyper-parameters $\alpha_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}} + N[x_i, x'_i|\mathbf{par}_{\boldsymbol{i}}]$ and $\tau_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}} + T[x_i|\mathbf{par}_{\boldsymbol{i}}]$. Thus the MAP parameter estimates are

$$\hat{q}_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}} = \frac{N[x_i, x'_i|\mathbf{par}_{\boldsymbol{i}}] + \alpha_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}}}{T[x_i|\mathbf{par}_{\boldsymbol{i}}] + \tau_{x_i,x'_i|\mathbf{par}_{\boldsymbol{i}}}} \ . \tag{86}$$

5.4.2 Structure Estimation

Estimating the CTBN structure could be accomplished by statistical tests of the independence of the processes. Yet, we are unaware of any methods that use this or of suitable independence tests.

Instead, CTBN structures have been estimated by graph scoring functions. If the score function decomposes as the likelihood does (Equation 83) into a sum of terms, one per variable, in which the selection of a variable's parents only affects the term for the same variable, the search for the maximal scoring graph is very simple. Each variable's parents can be chosen independently by maximizing the corresponding term in the sum. While there are an exponentially large (in the total number of variables) number of parent sets to consider for each variable, if we limit the cardinality of parent sets to no more than D, then each variable's parents can be chosen by exhaustive search and the total running time is $O(L2^D)$, which is linear in the number of variables, L.

This is in contrast to Bayesian networks where a similar strategy does not lead to an efficient algorithm (unless a variable ordering is known *a priori*). Learning CTBNs structure is efficient because there are no restrictions on the graph: A CTBN's graph may be cyclic. A similar situation arises with dynamic Bayesian networks (DBNs). If we only allow inter-time-slice edges (those from the "previous" time point to the "current" time point), the graph structure may be searched efficiently, like in CTBNs. However, if we allow intra-time-slice edges (those within the "current" time point) in a DBN, we must enforce acyclicity constraints and the search is no longer efficient.

The Bayesian information criterion (Lam & Bacchus, 1994) can be made into a score:

$$score_{BIC}(\mathcal{G} : \mathcal{Tr}) = \left(\sum_i l_{X_i}(\mathcal{Tr})\right) - \frac{\ln|\mathcal{Tr}|}{2} Dim(\mathcal{G}) \tag{87}$$

$$= \sum_i \left(l_{X_i}(\mathcal{Tr}) - \frac{\ln|\mathcal{Tr}|}{2} \left|\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}\right| \right) \tag{88}$$

where $Dim(\mathcal{G})$ is the number of independent parameters in the network defined by the graph $\mathcal{G}$ and $\left|\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_{\boldsymbol{i}}}\right|$ is the number of independent parameters in the conditional intensity matrices associated with X_i. This second term is equal to $n_i(n_i-1)$ (because the diagonal elements are not independent) times the number of parent instantiations. The data size, $|\mathcal{Tr}|$, is the number of transitions in the trajectory $\mathcal{Tr}$ (or in the total data set if it consists of multiple trajectories). This score is consistent (Nodelman et al., 2003) because the term $l_{X_i}(\mathcal{Tr})$ grows linearly with the amount of data and represents the likelihood and the second term grows logarithmically with the amount of data and penalizes excess parameters.

A Bayesian score can also be constructed by placing a prior on graphs (as well as parameters) and finding the maximum of $\ln P(\mathcal{G} \mid \mathcal{Tr}) = \ln p(\mathcal{Tr} \mid \mathcal{G}) + \ln P(\mathcal{G}) - \ln p(\mathcal{Tr})$. The last term isn't affected by the choice of $\mathcal{G}$, so we drop it. We assume structure modularity: $\ln P(\mathcal{G}) = \sum_i \ln P(\mathrm{Par}_i)$. The remaining data term, $\ln P(\mathcal{Tr} \mid \mathcal{G})$, is the (logarithm of the) integral of the likelihood multiplied by the prior, over all possible parameter values. Using the independent gamma priors from above, this decomposes into a separate term for each variable (dropping the $\ln P_0(\mathcal{Tr}(0))$ term which does not affect the choice of $\mathcal{G}$):

$$\ln p(\mathcal{Tr} \mid \mathcal{G}) = \sum_{i,\mathbf{par}_{\boldsymbol{i}},x_i\neq x_i'} \ln \int_0^\infty \frac{\left(\tau_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}\right)^{\alpha_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}+1}}{\Gamma(\alpha_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}+1)} \exp\Big[-(T[x_i|\mathbf{par}_{\boldsymbol{i}}] + \tau_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}) q_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}$$
$$+ (N[x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}] + \alpha_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}) \ln q_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}\Big] \, dq_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}} \tag{89}$$

$$= \sum_i \sum_{\mathbf{par}_{\boldsymbol{i}},x_i\neq x_i'} \ln \left(\frac{\left(\tau_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}\right)^{\alpha_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}+1}}{\Gamma(\alpha_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}+1)} \frac{\Gamma(N[x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}] + \alpha_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}} + 1)}{\left(T[x_i|\mathbf{par}_{\boldsymbol{i}}] + \tau_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}\right)^{N[x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}]+\alpha_{x_i,x_i'|\mathbf{par}_{\boldsymbol{i}}}+1}} \right) \tag{90}$$

$$= \sum_i lscore_B(\mathbf{par}_{\boldsymbol{i}} : \mathcal{Tr}) \tag{91}$$

where the last line is by the definition of $lscore_B$. This derivation is almost the same as the one given in Nodelman et al. (2003). The difference is that our prior consists of a gamma distribution for each independent variable whereas their prior consists of a gamma distribution for each diagonal rate parameter and a Dirichlet prior over the ratios $q_{x_i,x_i'|\mathbf{par}_i}/q_{x_i|\mathbf{par}_i}$. The two are equivalent, but parameterized differently.

The Bayesian score is therefore

$$score_B(\mathcal{G} : \mathcal{Tr}) = \sum_i lscore_B(\mathbf{par}_i : \mathcal{Tr}) + \ln P(\mathrm{Par}_i) \ . \tag{92}$$

It converges to the BIC score in the limit of infinite data (Nodelman et al., 2003) and is therefore also consistent.

5.4.3 Incomplete Data

For the case where the trajectory $\mathcal{Tr}$ is incomplete, we are back in the same situation as in Section 2.6. As the likelihood takes the same form in a CTBN as in a general CTMP, the solutions for maximizing the likelihood of this incomplete trajectory have the same form. Namely, if we compute the expected sufficient statistics (using inference), we can apply gradient ascent or expectation-maximization to find the maximum likelihood parameters. The gradient is

$$\frac{\partial p(\mathcal{Tr})}{\partial q_{x_i,x_i'|\mathbf{par}_i}} = p(\mathcal{Tr}) \left(\frac{\bar{N}[x_i, x_i'|\mathbf{par}_i]}{q_{x_i,x_i'|\mathbf{par}_i}} - \bar{T}[x_i|\mathbf{par}_i] \right) \tag{93}$$

and the expectation-maximization update equation is the same as Equation 85 except the sufficient statistics are replaced by their expected values, given the partially observed trajectory and the current model.

For graph estimation, we can apply structural expectation-maximization (Friedman, 1997) (SEM) to CTBNs (Nodelman, Shelton, & Koller, 2005). While SEM for Bayesian networks can be a little complex due to the structure search step, for CTBNs, it is simpler as the structure search step need not enforce acyclicity constraints and therefore can be carried out more simply (see above). The tricky point (which also holds for standard Bayesian networks) is that the graph search scoring function must be calculated using expected sufficient statistics and therefore, given a current model, our inference algorithm must produce expected sufficient statistics not only for the current model's parent sets, but also for any other parent sets to be considered by the structure search. If using exact inference (by flattening the CTBN to a general CTMP), these are available. However, approximate methods (below) differ in how simple it is to extract such expected sufficient statistics. Once this inference is performed, a joint optimization of parameters and structure is performed, the new model is used to find new expected sufficient statistics, and the process repeats.

5.5 Approximate Inference

As mentioned in Section 5.3, exact inference is intractable when there are many concurrently missing variables. Therefore, many approximate inference methods have been developed. We briefly cover them in this section, but would refer to the full papers for more complete descriptions.

5.5.1 Sampling-Based Inference

Sampling is an obvious method for producing approximate inference. It has a number of advantages. First, it produces a set of full trajectories from which any inference question can be answered. Second, most sampling methods converge to the correct value if allowed to run long enough. Third, sampling methods are usually easily parallelized, lending themselves to multiple processors and multiple cores.

Hobolth and Stone (2009) have a description of a number of such methods for the unstructured case and for full evidence at the beginning and the end of the trajectory, but no evidence in between. Here we discuss work on CTBNs and for more general evidence patterns.

Fan and Shelton (2008) and Fan, Xu, and Shelton (2010) developed an importance sampler and a particle filter and smoother. Forward sampling (like in Figure 18) can be turned into an importance sampler by taking the observed data as given and sampling for the missing portions, marching time along. The weight of the sample is the probability of having sampled the observed data (which was not sampled) given the trajectory up to those data. The problem arises when a variable goes from being not observed to being observed. In this case, the sampling must agree with "up coming" observation evidence. Adding a transition exactly when the evidence starts is not correct (as there is almost surely not an event at that time). These samplers handle it with some forward look ahead to sample the necessary transition in advance, with suitable importance weight corrections. This is then extended to a particle filter and smoother which are resampled based on the number of transitions, rather than the absolute time. This method was extended to more general temporal models by Pfeffer (2009).

El-Hay, Friedman, and Kupferman (2008) developed a Gibbs sampler for CTBN models. They start with a simply developed trajectory that agrees with all of the evidence. Then, the algorithm removes a single variable's full trajectory and resamples it (keeping any time periods during which the value is known). Conditioned on the full trajectories of a variable's Markov blanket (the union of the variable's children, parents, and children's parents), the trajectory for that variable is independent of all other variables, so the sampler needs to only consider the variable's Markov blanket. The posterior distributions over the times of transitions are no longer exponential distributions. Their forms are complex and thus the Gibbs sampler must sample by performing binary search.

Fan and Shelton (2009) combined the ideas from the Gibbs sampler and their earlier work on importance sampling to produce a Metropolis-Hastings sampler. The importance sampling method is used instead of Gibbs sampling and the importance weight is used to find the acceptance probability. While faster to generate samples, the samples take longer to converge. The balance of the trade-off depends on the typicality of the evidence and the inference query.

Rao and Teh (2011, 2013) used uniformization to develop an auxiliary Gibbs sampler which is faster than the previous Gibbs sampler. The auxiliary variables are the times of the self-transitions from an uniformization sampler (Section 2.3). Thus to resample a variable, the algorithm samples auxiliary times, given the old trajectory (which can be done quickly). It then throws away all of the transitions, but keeps the full set of times (old times and the new times). Then, using a forward-backward two-pass algorithm, state transitions are sampled from the uniformized discrete-time process (conditioned on the evidence). Finally, the self transitions are discarded. Rao and Teh (2012) extended this to a time-varying uniformization rate to speed up convergence, but only explicitly in the case of an unstructured process.

5.5.2 Non-Sampling Methods

A number of other non-sampling methods have also been proposed. Their advantages include determinism (often helpful when used inside of EM to keep the estimates consistent), and fewer parameters that need to be set well (number of samples, length of burn-in, and others).

Cohn, El-Hay, Kupferman, and Friedman (2009) and Cohn, El-Hay, Friedman, and Kupferman (2010) derived a mean-field approximation. The approximate distribution is an independent *time-inhomogeneous* Markov process for each variable. That is, the variables are independent (in the approximation), but the intensities depend on time. The natural parameterization also differs slightly. Instead of transition rates, transition densities are used, but the idea is the same. The resulting algorithm changes one variable's distribution at a time, again depending only on the Markov blanket. The update involves solving a system of differential equations (to get the time-varying parameters of the inhomogeneous Markov process). These are solved by adaptive integration which means that less computation is required during intervals of less rapid change. The result is that the time-varying parameters are represented by a series of time-value points (those produced during adaptive integration) and linear interpolation between these points.

Nodelman, Koller, and Shelton (2005) derived an expectation-propagation method. The propagation uses piece-wise constant time-homogeneous Markov processes, where each piece corresponds to a period of constant evidence. These piece-wise constant approximations are propagated instead of the true marginals (as such marginals would be intractably large). Saria, Nodelman, and Koller (2007) extended this method to subdivide the pieces of the approximations further and adaptively. El-Hay, Cohn, Friedman, and Kupferman (2010) produced a belief propagation algorithm in the same spirit as the mean-field approximation above, employing a free energy functional for CTMPs. Instead of propagating piece-wise time-homogeneous Markov processes, they propagate a single time-inhomogeneous Markov process and then use the same adaptive integration representation as in mean-field. The result is more adaptive and mathematically cleaner.

Finally, for filtering (but not general inference), Celikkaya, Shelton, and Lam (2011) developed a factored version of a uniformized Taylor expansion to approximate the matrix exponential calculations. The result is something similar to that of Boyen and Koller (1998) for dynamic Bayesian networks, but also involving a truncation of an infinite sum and a mixture of propagation distributions. This method is the only current non-sampling method with accuracy bounds, although they are very loose.

5.6 Extensions

As shown above, a CTBN can be converted into a sum of decision diagrams. In that way decision diagrams (and other similarly convertible models) can be viewed as extensions of

CTBNs. Many of the non-Markovian processes of Section 1.1 could, if restricted in the right way, also be so viewed. However, there are a few more direct extensions of CTBNs.

First, El-Hay et al. (2010) introduced continuous-time Markov networks (CTMNs). They are *undirected* graphical models of Markov processes in the same way that CTBNs are *directed* graphical models. They model the subclass of reversible processes, ones for which detailed balance holds: There exists a distribution π (the stationary distribution of the process) such that $\pi(\boldsymbol{x})q_{\boldsymbol{x},\boldsymbol{x'}} = \pi(\boldsymbol{x'})q_{\boldsymbol{x'},\boldsymbol{x}}$ for all pairs of states $\boldsymbol{x}$ and $\boldsymbol{x'}$. A CTMN can be converted to a CTBN by replacing each undirected edge with a pair of directed edges. Their parameterization directly reveals the stationary distribution of the process as a Markov network.

Second, Portinale and Codetta-Raiteri (2009) and Codetta-Raiteri and Portinale (2010) showed an extension of CTBNs to allow for simultaneous transition of multiple variables. It is based on Petri nets and encodes "cascades" of transitions that all happen simultaneously.

Finally, Weiss, Natarajan, and Page (2012) presented a method for constructing the local rate matrices for each variable not as a matrix, but as the multiplication of regression trees. This is akin to exploiting context-specific independence (Shimony, 1991) in a standard Bayesian network by use of trees (Boutilier, Friedman, Goldszmidt, & Koller, 1996). This multiplication of trees is not the same as our reduction of a CTBN to a sum of EV*MDDs (see Figure 17). In particular, their trees do not require the tests be made in a particular variable order, they use trees instead of DAGS, and they multiply the trees together (instead of adding them). Weiss et al. (2012) also give a boosting-style algorithm for learning this parameterization. No similar method is known for learning a sum of EV*MDDs.

6. Applications and Current Directions

To provide some context for the theory and algorithms above, we describe how these methods have been used in applications. We then discuss what we believe to be the most promising and pressing research directions.

6.1 Decision-Diagram-Based Models

Structured CTMPs arise in many applications areas, from performance and reliability evaluation of computer systems to the investigation of biological systems. As the underlying CTMPs describing the dynamics being analyzed are usually very large, most software tools used for such studies rely on compact symbolic techniques to encode them.

In particular, PRISM (Kwiatkowska et al., 2011) uses a hybrid form of MTBDDs, Möbius (Deavours et al., 2002) uses Matrix Diagrams (a data structure almost equivalent to EV*MDDs), and SMART (Ciardo et al., 2006) uses EV*MDDs to encode the transition rate matrix of a CTMP. These tools can compute both stationary and transient exact numerical solutions of compactly encoded CTMPs. (Indeed, they can compute much more complex *stochastic temporal logic properties* such as those that can be expressed in CSL (Baier, Haverkort, Hermanns, & Katoen, 2000), but these, too, ultimately require a sequence of stationary or transient numerical solutions.) While such exact solutions place very large computational demands due to the exponential explosion of the state space, the situation is often somewhat mitigated by the fact that, in most applications targeted by these tools, the actual state space is a small subset of the full cross-product of the state variable values.

As an example application, we briefly summarize a study done using PRISM for the analysis of a complex biological pathway called FGF (Fibroblast Growth Factor) (Heath, Kwiatkowska, Norman, Parker, & Tymchyshyn, 2006). The state of the system consists of the number of proteins (e.g., A, B) or protein complexes (e.g., $A{:}B$) present at the current time. The events in the system consist of various reactions such as *complexation* (e.g., $A + B \rightarrow A : B$) and *decomplexation* (its reverse, $A{:}B \rightarrow A + B$), as well as *degradation* (e.g., $A \rightarrow$). Finally, each event has an occurrence rate, which can of course depend on the number of proteins currently present for the types involved in the particular reaction. The actual model for the FGF pathway, even after substantial simplifications to focus only on key and well-known aspects in real cells, contains 87 different proteins or protein complexes (each of them corresponding to a local state variable) and 50 different reactions (if we count complexation and decomplexation separately). We stress that each reaction concerns just a few proteins or compounds; thus its decision diagram representation in isolation is quite compact.

Even the smallest meaningful model where there is only zero or one protein or compound of each type would have a potential state space of size 2^{87}. However, as almost always the case in this type of models, only a tiny fraction of these states is reachable, thus the model used in the work of Heath et al. (2006) has merely 801,616 states and 560,000 state-to-state transitions. The study focused on several key questions such as what fraction of time particular proteins are bound, or the probability that a particular degradation has occurred within a given time bound, all quantities obtainable through numerical stationary or transient analysis of the underlying CTMP. Notwithstanding the relatively small size of the state space (which could likely be scaled by a factor of 1000, to around 10^8 states, given a modern workstation with sufficient memory) the results and predictions obtained from this model using PRISM were shown to agree with biological data, demonstrating the viability of these technique to perform "in silico genetics" as a much less costly alternative to the "in vitro" experiments traditionally performed in biology.

6.2 Continuous-Time Bayesian networks

CTBNs have been employed on a number of real-world datasets and problems including life event history data (Nodelman et al., 2003), user activity modeling (Nodelman & Horvitz, 2003), computer system failure modeling (Herbrich, Graepel, & Murphy, 2007), mobile robotics (Ng, Pfeffer, & Dearden, 2005), network intrusion detection (Xu & Shelton, 2008, 2010), phylogentic trees (Cohn et al., 2009), social networks (Fan & Shelton, 2009), cardiovascular health model (Weiss et al., 2012), and heart failure (Gatti, Luciani, & Stella, 2012). Many of these also innovated in extending the CTBN framework. For instance, Ng et al. (2005) allowed for continuous-state variables whose dynamics are dictated by differential equations. The form of evidence is more limited, but a particle filter is developed for this situation. Cohn et al. (2009) applied the CTBN model to a "time-tree" to allow branching (as first done in Felsenstein, 1981 for general CTMPs). Weiss et al. (2012) added context-specific independence.

To give an idea of the application of CTBNs, we briefly review the intrusion detection work of Xu and Shelton (2008, 2010). In this work, the goal was to build a model of normal

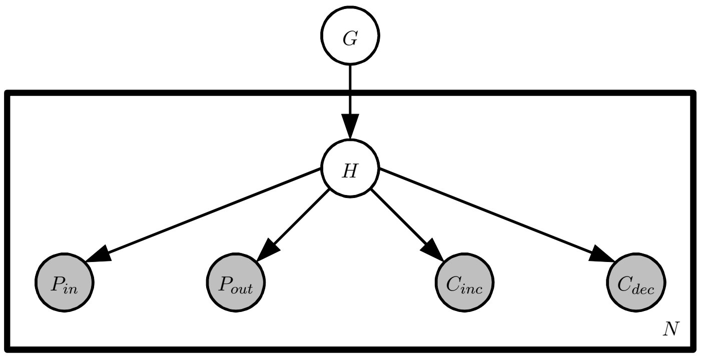


Figure 19: CTBN model for network traffic (Xu & Shelton, 2010). N is the number of destination ports.

computer system events, specific to a particular machine. This model could then be used to detect abnormal events or time windows as a method of intrusion detection.

Two models were built: one modeling network traffic in and out of the machine, and one modeling system calls from processes. The network model was as in Figure 19. There is one global hidden variable G with four states. The traffic is divided into different destination ports (for instance, 80 for HTTP traffic and 995 for POP traffic). The most frequent eight ports are separated out and the traffic from the remaining destinations are grouped together. Each of these $N = 9$ groups has its own model (the plate in Figure 19). This submodel has one hidden variable H and four completely observed binary variables, P_{in}, P_{out}, C_{inc}, and C_{dec}, representing packets sent and received and connections started and stopped respectively. These observed variables toggle state to represent an event of the relevant type, but their state has no intrinsic meaning. Therefore, their matrices have only a single independent parameter: the rate of transition from either state to the other. In this way, these observed variables are really conditional Poisson processes.

The hidden variable H is structured to exploit domain knowledge. It has 8 states which are grouped into pairs, one pair for each of its children. For each pair only one of its children has a non-zero rate. Thus, H encodes what type of event will happen and some substate of this meta-state.

The entire model has 4×8^9 or approximately 500 million hidden states (as the observed variables are observed at all times, only the distribution over the G and the Hs need be tracked). Yet, each submodel has only 8 hidden states, so exact inference over a submodel is very reasonable. Thus, they adapted the particle filter and smoother of Fan and Shelton (2008) to distributional particles, producing a Rao-Blackwellized particle filter in which G is sampled and each of the models (which are independent, given a full trajectory of G) are reasoned about exactly.

This inference method allows learning of specific models to each host using EM. These models were then run on data in which computer virus or worm traffic had been injected (very slowly to make it blend in with background traffic). The model was asked the likelihood of 50-second window of traffic (given the previous traffic). This likelihood was thresholded to produce an alarm (if the likelihood dropped too low). The results out-performed SVM-spectrum kernels, nearest-neighbor (using features from the computer network literature for this task), and other methods on this task and other similar tasks.

For process system calls, the model was similar, with a single hidden variable coordinating the behavior of a set of observed system-call variables. The dataset on which this model was trained had time stamps for each system call. However, due to clock resolution, many time stamps were the same. Yet, the temporal order was preserved (although the exact durations between events was not). The paper demonstrates a method to use such data, without assuming event durations, but employing the ordering. In this case, the results were better than the SVM-spectrum kernel and nearest-neighbor and the same as stide with frequency thresholding (Warrender, Forrest, & Pearlmutter, 1999).

6.3 Relative Comparison

Neither of the above two applications could currently be tackled with the other modeling language. In the biological pathway example of Section 6.1, a transition in the system involved more than one variable (increasing the number of protein complexes, while decreasing the number of individual proteins). A CTBN cannot represent simultaneous transitions of multiple variables. The pathways cannot be reformulated in terms of composite variables to prevent such simultaneous transitions without placing all of the state in a single variable.

As a simpler example, consider a system of three variables, x, y, and z. A single event performs three variable updates at the same time: $\{x' \Leftarrow x+y; y' \Leftarrow y+1; z' \Leftarrow z-1\}$ with rate $r(x,y,z)$. We assume that each variable is a natural number in the range $[0,\ldots,n]$ and that any update that would move a variable outside its range is disabled. Figure 20 demonstrates how an EV*MDD can encode such a transition. Neither a Kronecker encoding nor a CTBN can encode this event without merging variables.

Likewise, the network traffic example from Section 6.2 cannot be handled with current decision-diagram-based models. It depends on hidden unseen variables and estimation of transient solutions conditioned on data. More critically, it relies on estimation of the model parameters from data, which has not been developed for decision-diagram models.

6.4 Current Research Directions

There are a range of open modeling, algorithmic, and theoretic problems. First, questions of steady-state distributions and efficient exact solutions have not been addressed for CTBNs (as they have for EV*MDDs). Similarly, questions of structure and parameter estimation and approximate inference have not been addressed for EV*MDDs (as they have for CTBNs).

Optimal decision making has been formulated as general continuous-time Markov decision processes (Puterman, 1994). Yet, extending the general mathematical framework to structured variable-based models is largely unexplored (Kan & Shelton, 2008).

CTBNs were extended to handle continuous-valued variables and measurements in a limited fashion (Ng et al., 2005), but otherwise this has been unexplored. For many applications this is critical. If the underlying system is discrete and the measurements are continuous, techniques like those in Section 2.5 work. But, systems with continuous state require stochastic differential equations (Øksendal, 2003), at least in some form. The work

x	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
y	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
z	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
x'	–	–	–	–	–	–	–	–	–	0	1	2	1	2	–	–	–	–	0	1	2	1	2	–	–	–	–
y'	–	–	–	–	–	–	–	–	–	1	1	1	2	2	–	–	–	–	1	1	1	2	2	–	–	–	–
z'	–	–	–	–	–	–	–	–	–	0	0	0	0	0	–	–	–	–	1	1	1	1	1	–	–	–	–

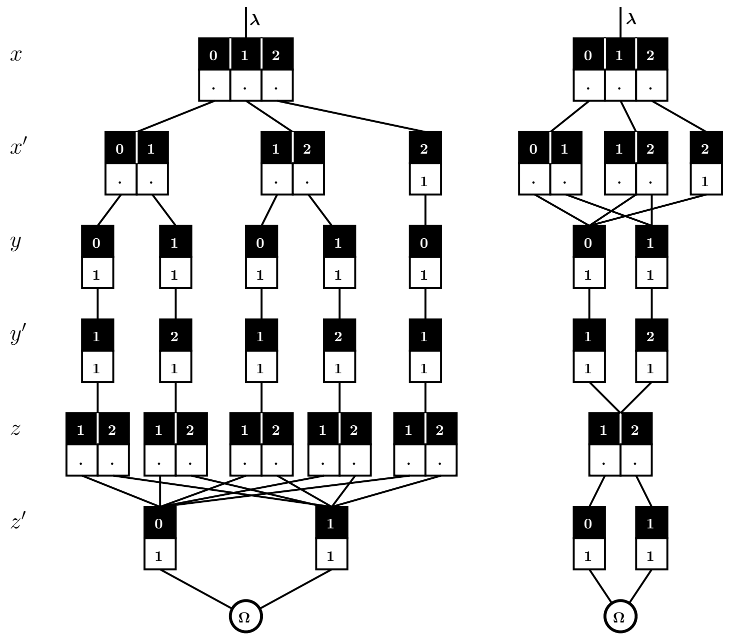


Figure 20: An example of an EV*MDD encoding simultaneous transitions of multiple variables. Top: the 10 possible transitions and resulting states (dash indicates disabled) from state (x, y, z) to state (x', y', z'). Bottom, left: worst case for an arbitrary set of 10 rates for $r(x, y, z)$. Bottom, right: best case when $r(x, y, z) = r_1(x, y)r_2(z)$. A dot indicates a positive value (used to encode the particular rates). In each block of dots, one must be equal to 1 and the others must be ≤ 1.

of Särkkä (2006) describes filtering and smoothing in such models. Yet, parameter estimation is much more difficult and systems with both continuous and discrete state variables have not been systematically addressed.

Finally, new approximate inference methods are always of interest (as with any probabilistic model). Recent methods such as the auxiliary Gibbs sampler (Rao & Teh, 2013) and belief propagation (El-Hay et al., 2010) demonstrate that exploiting the properties of continuous time can lead to great benefits. We hope that further research explores more such methods.

7. Conclusions

Compared with discrete-time models, CTMPs are better suited for domains in which data have real-valued time stamps (the time between events is not regular or well-approximated by a single "clock step rate"). Thus, in selecting a value for the time-slice-width (Δt) for a discrete-time model, either the time width will be large resulting in multiple events per time window (obscuring temporal information), or it will be small resulting in unnecessary computational burdens (propagating across many time windows). Further, the optimal middle ground between too large and too small will differ depending on the data size, the application, and the model component.

We have presented two different CTMP modeling languages. Edge-valued decision diagrams (of Section 4) are more general: They allow multiple variables to change simultaneously. By contrast, CTBNs directly encode independence assumptions (see Section 5). Either an EV*MDD or a CTBN might be more compact for a given situation, although any CTBN can be compactly rerepresented as a *sum* of EV*MDDs (see Section 5.1.3).

The models' forms and histories have given rise to differences in available algorithms, and here the distinctions are greater. The literature on exact solution methods is richer for decision diagram models. Furthermore, this literature is more focused on computing the model's steady state. Approximate methods (especially for transients) and model estimation are notably absent (from an artificial intelligence point-of-view). The literature for CTBNs is more focused on model estimation and approximate inference conditioned on evidence. The CTBN literature has paid no attention to issues of reachability (when much of the joint state space is not reachable) and optimization of exact inference methods.

For processes with a natural synchronization clock (such as modeling daily high and low temperatures), a discrete-time model is the best fit. For processes without such a natural time-slice-width we recommend a continuous-time model. If the questions of interest are about steady states of the system or an exact solution is necessary, an EV*MDD is probably the best choice. If the model must be built from data or approximate inference (especially conditioned on data) is necessary, a CTBN is probably the best choice.

However, we have shown that the two models share much in common. Thus, we hope that the efficient exact algorithms from EV*MDDs can be applied to CTBNs and the approximate inference and model estimation methods from CTBNs can be applied to EV*MDDs. If so, then the choice of model would depend more upon the model properties and not the existing suite of algorithms. In particular, a CTBN makes the assumption that each variable is distinct. In contrast, a disjunctive EV*MDD encoding decomposes the system into local events. The variable-level independencies are more easily read from a CTBN graph, but they disallow simultaneous transition of multiple variables. The application domain should guide whether variable-level explicit independences or simultaneous transitions are more important.

Regardless of the model used, we believe time is a continuous quantity and best modeled as such. While the introduction of the matrix exponential would at first seem to complicate matters (compared with discrete time), we believe it makes the true coupling of variables more obvious and opens up mathematical and algorithmic possibilities for more efficient and precise solutions.

Acknowledgements

Shelton was supported by DARPA award FA8750-14-2-0010 and by The Laura P. and Leland K. Whittier Virtual PICU at Children's Hospital Los Angeles (awards UCR-12101024 and 8220-SGNZZ0777-00). Ciardo was supported by the NSF through grant CCF-1442586.

Appendix A. NP-hardness of CTBN Inference

The theorem and proof of the NP-hardness of CTBN inference are straight-forward extensions of the similar proof for Bayesian networks (Koller & Friedman, 2009). The literature has widely accepted it to be true, but no proof has been formally presented. Thus, while straight-forward, we present it here for completeness.

Definition 1. CTBN-Inf *is the following decision problem. Given a CTBN specified as a directed graph $\mathcal{G}$ over nodes $\{X_1, X_2, \ldots, X_L\}$, a set of conditional intensity matrices $\mathcal{Q} = \{\boldsymbol{Q}_{\boldsymbol{X_i}|\mathbf{par}_i}\}$, and an initial distribution π in which each node is independent with marginals $\{\pi_{X_i}\}$; a variable X_j; a value for X_j, x_j; and a time $t > 0$, decide whether $P_{\mathcal{G},\mathcal{Q},\pi}(X_j(t) = x_j) > 0$.*

Theorem 1. ***CTBN-Inf*** *is NP-hard.*

Proof. The proof is a polynomial time reduction from 3-SAT, following the same lines as the similar proof for Bayesian networks.

Given a 3-SAT problem with variables $z_1, z_2, \ldots, z_m$ and clauses $c_1, c_2, \ldots, c_k$ in which $z_{A(i,j)}$ is the jth variable ($j \in \{1, 2, 3\}$) in clause i ($i \in \{1, 2, \ldots, k\}$) with sign $s_{A(i,j)} \in \{+, -\}$, we construct a CTBN with $m + 2k - 1$ binary variables (taking values F or T): $Y_1, Y_2, \ldots, Y_m$, $C_1, C_2, \ldots, C_k$, $B_1, B_2, \ldots, B_{k-2}$, and S.

Variable Y_i has no parents, a uniform initial distribution π_{Y_i}, and an intensity matrix $\boldsymbol{Q}_{\mathbf{Y}_i|\emptyset}$ that is all 0.

Variable C_i has three parents: $Y_{A(i,1)}$, $Y_{A(i,2)}$, and $Y_{A(i,3)}$. If none of the truth value of the parents $(y_{A(i,1)}, y_{A(i,2)}, y_{A(i,3)})$ match the formula's signs $(s_{A(i,1)}, s_{A(i,2)}, s_{A(i,3)})$, the conditional intensity matrix is all 0. For the other parent assignments (in which at least one variable matches), the conditional intensity matrices are $\begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$. The initial distribution is $\begin{bmatrix} 1 & 0 \end{bmatrix}$.

Variable B_1 has parents C_1 and C_2. Variable B_i (for $1 < i < k - 1$) has parents B_{i-1} and C_{i+1}. Variable S has parents B_{k-2} and C_k. For all of these variables, the conditional intensity matrix if the two parents' values are T is $\begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$. Otherwise, the conditional intensity matrix is all 0. All of these variables have an initial distribution of $\begin{bmatrix} 1 & 0 \end{bmatrix}$.

This reduction is polynomial in size (all numeric values are small and there are a polynomial number of variables, each with a maximum of 3 parents) and can obviously be output in polynomial time. By construction, Y_i selects at time 0 a truth value for z_i and never changes. Each C_j will then eventually change to T if and only if the clause is satisfied by the selected truth values. B_j will eventually change to be T if and only if clauses 1 through $j + 1$ are all T. S will similarly eventually change to be T if and only if all clauses are satisfied.

Because of the Markov nature of the process, for any time $t > 0$, $P_{\mathcal{G},\mathcal{Q},\pi}(S(t) = T) > 0$ if the formula is satisfiable and the same probability is 0 if the formula is not satisfiable. $\square$

This demonstrates that determining whether a marginal is non-zero is NP-hard. By similar construction as for Bayesian networks (Koller & Friedman, 2009), this can be extended to show that absolute and relative error formulations of inference are also NP-hard.

References

Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., & Franceschinis, G. (1995). *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons.

Asmussen, S., Nerman, O., & Olsson, M. (1996). Fitting phase-type distributions via the EM algorithm. *Scandavian Journal of Statistics*, *23*, 419–441.

Baier, C., Haverkort, B. R., Hermanns, H., & Katoen, J.-P. (2000). Model checking continuous-time Markov chains by transient analysis. In *Proceedings of Computer Aided Verification*, pp. 358–372.

Baskett, F., Chandy, K. M., Muntz, R. R., & Palacios-Gomez, F. (1975). Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, *22*(2), 335–381.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, pp. 115–123.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 33–42.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Buchholz, P., Ciardo, G., Donatelli, S., & Kemper, P. (2000). Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, *12*(3), 203–222.

Burch, J. R., Clarke, E. M., & Long, D. E. (1991). Symbolic model checking with partitioned transition relations. In *Int. Conference on Very Large Scale Integration*, pp. 49–58. IFIP Transactions, North-Holland.

Celikkaya, E. B., Shelton, C. R., & Lam, W. (2011). Factored filtering of continuous-time systems. In *Proceedings of the Twenty-Seventh International Conference on Uncertainty in Artificial Intelligence.*

Ciardo, G., Jones, R. L., Miner, A. S., & Siminiceanu, R. (2006). Logical and stochastic modeling with SMART. *Performance Evaluation*, *63*, 578–608.

Ciardo, G., & Siminiceanu, R. (2002). Using edge-valued decision diagrams for symbolic generation of shortest paths. In *Proceedings of Formal Methods in Computer-Aided Design (FMCAD)*, LNCS 2517, pp. 256–273. Springer.

Ciardo, G., & Yu, A. J. (2005). Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Proceedings of Correct Hardware Design and Verification Methods (CHARME)*, LNCS 3725, pp. 146–161. Springer.

Ciardo, G., Zhao, Y., & Jin, X. (2012). Ten years of saturation: a Petri net perspective. *Transactions on Petri Nets and Other Models of Concurrency*, *V*, 51–95.

Clarke, E., Fujita, M., McGeer, P. C., Yang, J. C.-Y., & Zhao, X. (1993). Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. In *IWLS '93 International Workshop on Logic Synthesis*.

Codetta-Raiteri, D., & Portinale, L. (2010). Generalized continuous time Bayesian networks and their GSPN semantics. In *European Workshop on Probabilistic Graphical Models*, pp. 105–112.

Cohn, I., El-Hay, T., Friedman, N., & Kupferman, R. (2010). Mean field variational approximation for continuous-time Bayesian networks. *Journal of Machine Learning Research*, *11*(Oct), 2745–2783.

Cohn, I., El-Hay, T., Kupferman, R., & Friedman, N. (2009). Mean field variational approximation for continuous-time Bayesian networks. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*(3), 142–150.

Deavours, D. D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J. M., Sanders, W. H., & Webster, P. G. (2002). The möbius framework and its implementation. *IEEE Transactions on Software Engineering*, *28*(10), 956–969.

Didelez, V. (2008). Graphical models for marked point processes based on local independence. *Journal of the Royal Statitical Society: Series B*, *70*(1), 245–264.

Donatelli, S. (1994). Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proceedings of International Conference on Application and Theory of Petri Nets (ICATPN)*, LNCS 815, pp. 258–277. Springer.

El-Hay, T., Cohn, I., Friedman, N., & Kupferman, R. (2010). Continuous-time belief propagation. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 343–350, Haifa, Israel.

El-Hay, T., Friedman, N., & Kupferman, R. (2008). Gibbs sampling in factorized continuous-time Markov processes. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 169–178.

Fan, Y., & Shelton, C. R. (2008). Sampling for approximate inference in continuous time Bayesian networks. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics*.

Fan, Y., & Shelton, C. R. (2009). Learning continuous-time social network dynamics. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*.

Fan, Y., Xu, J., & Shelton, C. R. (2010). Importance sampling for continuous time Bayesian networks. *Journal of Machine Learning Research*, *11*(Aug), 2115–2140.

Felsenstein, J. (1981). Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, *17*, 368–376.

Fernandes, P., Plateau, B., & Stewart, W. J. (1998). Efficient descriptor-vector multiplication in stochastic automata networks. *Journal of the ACM*, *45*(3), 381–414.

Fox, B. L., & Glynn, P. W. (1988). Computing poisson probabilities. *Communications of the ACM*, *31*(4), 440–445.

Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 125–133.

Gatti, E., Luciani, D., & Stella, F. (2012). A continuous time Bayesian network model for cardiogenic heart failure. *Flexible Services and Manufacturing Journal*, *24*(4), 496–515.

Grassmann, W. K. (1977). Transient solutions in Markovian queueing systems. *Computers & Operations Research*, *4*(1), 47–53.

Gunawardana, A., Meek, C., & Xu, P. (2012). A model for temporal dependencies in event streams. In *Advances in Neural Information Processing Systems*, Vol. 24.

Heath, J., Kwiatkowska, M., Norman, G., Parker, D., & Tymchyshyn, O. (2006). Probabilistic model checking of complex biological pathways. In Priami, C. (Ed.), *Proceedings Computational Methods in Systems Biology (CMSB)*, Vol. 4210 of *Lecture Notes in Bioinformatics*, pp. 32–47. Springer Verlag.

Herbrich, R., Graepel, T., & Murphy, B. (2007). Structure from failure. In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, pp. 1–6. USENIX Association.

Hobolth, A., & Stone, E. A. (2009). Simulation from endpoint-conditioned continuous-time Markov chains on a finite state space, with applications to molecular evolution. *The Annals of Applied Statistics*, *3*(3), 1204–1231.

Kam, T., Villa, T., Brayton, R. K., & Sangiovanni-Vincentelli, A. (1998). Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, *4*(1–2), 9–62.

Kan, K. F., & Shelton, C. R. (2008). Solving structured continuous-time Markov decision processes. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics*.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., & Qadeer, S. (Eds.), *Proceedings of Computer Aided Verification*, Vol. 6806 of *LNCS*, pp. 585–591. Springer.

Kwiatkowska, M. Z., Norman, G., & Parker, D. (2004). Probabilistic symbolic model checking with PRISM: a hybrid approach. *Software Tools for Technology Transfer*, *6*(2), 128–142.

Lam, W., & Bacchus, F. (1994). Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, *10*, 269–293.

Moler, C., & Loan, C. V. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, *45*(1), 3–49.

Najfeld, I., & Havel, T. F. (1994). Derivatives of the matrix exponential and their computation. Tech. rep. TR-33-94, Center for Research in Computing Technology, Harvard University.

Najfeld, I., & Havel, T. F. (1995). Derivatives of the matrix exponential and their computation. *Advances in Applied Mathematics*, *16*, 321–375.

Ng, B., Pfeffer, A., & Dearden, R. (2005). Continuous time particle filtering. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1360–1365.

Nodelman, U., & Horvitz, E. (2003). Continuous time Bayesian networks for inferring users' presence and activities with extensions for modeling and evaluation. Tech. rep. MSR-TR-2003-97, Microsoft Research.

Nodelman, U., Koller, D., & Shelton, C. R. (2005). Expectation propagation for continuous time Bayesian networks. In *Proceedings of the Twenty-First International Conference on Uncertainty in Artificial Intelligence*, pp. 431–440.

Nodelman, U., Shelton, C. R., & Koller, D. (2002). Continuous time Bayesian networks. In *Proceedings of the Eighteenth International Conference on Uncertainty in Artificial Intelligence*, pp. 378–387.

Nodelman, U., Shelton, C. R., & Koller, D. (2003). Learning continuous time Bayesian networks. In *Proceedings of the Nineteenth International Conference on Uncertainty in Artificial Intelligence*, pp. 451–458.

Nodelman, U., Shelton, C. R., & Koller, D. (2005). Expectation maximization and complex duration distributions for continuous time Bayesian networks. In *Proceedings of the Twenty-First International Conference on Uncertainty in Artificial Intelligence*, pp. 421–430.

Øksendal, B. (2003). *Stochastic Differential Equations: An Introduction with Applications* (Sixth edition). Springer-Verlag.

Parikh, A. P., Gunamwardana, A., & Meek, C. (2012). Cojoint modeling of temporal dependencies in event streams. In *UAI Bayesian Modelling Applications Workshop*.

Pfeffer, A. (2009). CTPPL: A continuous time probabilistic programming language. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, pp. 1943–1950.

Plateau, B. (1985). On the stochastic structure of parallelism and synchronisation models for distributed algorithms. In *Proceedings of ACM SIGMETRICS*, pp. 147–153.

Portinale, L., & Codetta-Raiteri, D. (2009). Generalizing continuous time Bayesian networks with immediate nodes. In *Proceedings of the Workshop on Graph Structure for Knowledge Represetnation and Reasoning*, pp. 12–17.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C* (Second edition). Cambridge University Press.

Puterman, M. L. (1994). *Markov Decision Processes.* Wiley-Interscience.

Rajaram, S., Graepel, T., & Herbrich, R. (2005). Poisson networks: A model for structured point processes. In *Proceedings of the AI STATS 2005 Workshop.*

Rao, V., & Teh, Y. W. (2011). Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In *Proceedings of the Twenty-Seventh International Conference on Uncertainty in Artificial Intelligence.*

Rao, V., & Teh, Y. W. (2012). MCMC for continuous-time discrete-state systems. In *Advances in Neural Information Processing Systems 25*, pp. 710–718.

Rao, V., & Teh, Y. W. (2013). Fact MCMC sampling for Markov jump processes and extensions. *Journal of Machine Learning Research*, *1*, 1–26.

Roux, P., & Siminiceanu, R. (2010). Model Checking with Edge-valued Decision Diagrams. In *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010), NASA/CP-2010-216215*, pp. 222–226. NASA.

Saria, S., Nodelman, U., & Koller, D. (2007). Reasoning at the right time granularity. In *Proceedings of the Twenty-third Conference on Uncertainty in AI*, pp. 421–430.

Särkkä, S. (2006). *Recursive Bayesian Inference on Stochastic Differential Equations.* Ph.D. thesis, Helsinki University of Technology.

Shimony, S. E. (1991). Explanation, irrelevance and statistical independence. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 482–487.

Wan, M., Ciardo, G., & Miner, A. S. (2011). Approximate steady-state analysis of large Markov models based on the structure of their decision diagram encoding. *Performance Evaluation*, *68*, 463–486.

Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy, IEEE Computer Society.*

Weiss, J. C., Natarajan, S., & Page, D. (2012). Multiplicative forests for continuous-time processes. In *Advanced in Neural Information Processing Systems.*

Weiss, J. C., & Page, D. (2013). Forest-based point processes for event prediction from electronic health records. In *Proceedings of the European Conference in Machine Learning and Principals and Practice of Knowledge and Discovery in Databases (ECML-PKDD).*

Williams, C. K. I. (1998). Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In Jordan, M. I. (Ed.), *Learning in Graphical Models*, pp. 599–621.

Xu, J., & Shelton, C. R. (2008). Continuous time Bayesian networks for host level network intrusion detection. In *European Conference on Machine Learning*, pp. 613–627.

Xu, J., & Shelton, C. R. (2010). Intrusion detection using continuous time Bayesian networks. *Journal of Artificial Intelligence Research*, *39*, 745–774.

Appendix B

Deterministic Anytime Inference for Stochastic Continuous-Time Markov Processes

Appeared as

E. Busra Celikkaya and Christian R. Shelton. Deterministic Anytime Inference for Stochastic Continuous-Time Markov Processes. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML)*, 2014.

Deterministic Anytime Inference for Stochastic Continuous-Time Markov Processes

E. Busra Celikkaya CELIKKAE@CS.UCR.EDU
University of California, Riverside

Christian R. Shelton CSHELTON@CS.UCR.EDU
University of California, Riverside

Abstract

We describe a deterministic anytime method for calculating filtered and smoothed distributions in large variable-based continuous time Markov processes. Prior non-random algorithms do not converge to the true distribution in the limit of infinite computation time. Sampling algorithms give different results each time run, which can lead to instability when used inside expectation-maximization or other algorithms. Our method combines the anytime convergent properties of sampling with the non-random nature of variational approaches. It is built upon a sum of time-ordered products, an expansion of the matrix exponential. We demonstrate that our method performs as well as or better than the current best sampling approaches on benchmark problems.

1. Continuous-time stochastic systems

Continuous-time discrete-state stochastic models describe systems in which event times are not synchronized with a global clock. Examples include web searches (Gunawardana et al., 2012), computer networks (Xu & Shelton, 2010), social networks (Fan & Shelton, 2009), robotics (Ng et al., 2005), system verification (Baier et al., 2003), and phylogenetic trees (Cohn et al., 2009), among others. Discretizing time can be computationally expensive. The "time-slice" width must be much smaller than the shortest time between events. This can lead to inefficient computations during times in which events or expected events are less frequent. Much as the abstraction of real-valued numbers (and their implementation in floating-point rather than fixed-point representations) is helpful in the development of numeric algorithms, continuous-time is useful for stochastic dynamics systems.

Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

This paper focuses on Markovian models. In a discrete-time Markov process, given a row-stochastic matrix M and a distribution v (as a row-vector), the computation of $v_n = vM^n$ propagates v forward n time steps. In a continuous-time Markov process (CTMP), given a rate (intensity) matrix Q, $v_t = ve^{Qt}$ propagates v forward t time units in the same fashion. This is the critical computation step in filtering, smoothing, and parameter estimation. We focus on how to compute this *matrix exponential* when the size of v is very large and both v and Q have structure (allowing their efficient representation).

Except for the most trivial of cases, v_t has no internal structure. In particular, assume that the state space is factored, that is composed of joint assignments to state variables. Even if v is completely independent, v_t no longer has any structure (unless Q also represents a completely independent system). This is the same problem that arises in dynamic Bayesian networks (DBNs) in which forward propagation causes all variables in the system to be coupled. We assume that a full distribution over the state space is too large to be stored, and therefore seek an approximation.

1.1. Previous work

This problem has received attention in the verification literature for decision-diagram-based representations of the intensity matrix Q. However, the assumption behind this literature is that while Q may have structure to keep it representable, an exact answer is desired and therefore v_t is represented as a full vector. The shuffle algorithm is one such example (Fernandes et al., 1998).

By contrast, we assume that representing v_t explicitly is not possible. We would like to calculate expectations with respect to the distribution v_t. In our approach, we concentrate on continuous-time Bayesian networks (Nodelman et al., 2002) (CTBNs), but the method is general to any Q that is the sum of Kronecker products. Even the simplest expectations (like marginals) are NP-hard to compute (the proof is a straightforward extension of the proof for general Bayesian networks), so we focus on approximations. In the literature on CTBNs, there are a number of such methods that fall roughly into two groups. The first are variational approaches such as expectation propagation (El-Hay et al., 2010) and mean field (Cohn et al., 2009). These methods are deterministic. However, they do not converge to the true value as computation time increases and generally can only compute marginals or similar expectations. The second group are sampling approaches including importance sampling (Fan et al., 2010) and Gibbs sampling (Rao & Teh, 2011). These approaches converge to the true value and can estimate any expectation of v_t. However, they are random and this can cause problems when used inside other algorithms (like expectation-maximization).

1.2. Our approach

Our proposed method is deterministic and converges in the limit of infinite computational time. It can be viewed as a bridge between sampling and deterministic methods. We decompose the system into two pieces: a system (A) of completely independent components, and a correction (B). We reason exactly about the system A and add increasing number of correction terms derived from B. We generate a computation tree and traverse it using a priority queue, to select the larger correction terms earlier.

We first present our approach assuming that Q and probability vectors can be stored exactly. Then, we demonstrate how the calculations can be carried out efficiently when Q is structured. In section 2.4, we present a simple example to ground the derivation. Finally we demonstrate results comparing the computational efficiency of our method to other anytime convergent methods.

2. Matrix exponential calculation

Consider a CTMP with discrete states which is described by an initial state distribution row-vector v of size n and a rate matrix Q of size n-by-n. The rate matrix represents the rates by which the system transitions between states. The rate of transitioning from state i to j is $q_{ij} \geq 0$ and the rate of transitioning out of state i is $q_i = \sum_{j \neq i} q_{ij}$. The diagonal elements of the rate matrix are the negative row sums: $q_{ii} = -q_i$.

As stated above, the distribution at time t, for $t \geq 0$, is calculated by $v_t = ve^{Qt}$ where e^{Qt} is the matrix exponential with Taylor expansion $e^{Qt} = \sum_{k=0}^{\infty} \frac{1}{k!}(Qt)^k$. The matrix exponential calculation is very widely used in applied mathematics and there are many numerical methods to solve it (Moler & Loan, 2003).

If the state-space is structured as joint assignments to m variables, its size, n, grows exponentially with the number of variables, m. This makes the e^{Qt} calculation intractable for large systems. Using the structure of Q for this calculation is not straightforward because it is not preserved by the matrix exponential. Additionally, the commutative property does not hold for matrix exponential in general: For any same-sized matrices A and B, $e^{(A+B)t} \neq e^{At}e^{Bt} \neq e^{Bt}e^{At}$, unless the commutator $[A, B] = AB - BA$ vanishes. One possible decomposition comes from the Kronecker sum property: $e^{(A \oplus B)} = e^A \otimes e^B$. Yet, Kronecker sums alone can only describe rate matrices for systems in which all variables are independent.

For the general case, $e^{(A+B)t}$ can be seen as a perturbation of e^{At} in the direction of Bt (Najfeld & Havel, 1995) and can be represented as

$$e^{(A+B)t} = e^{At} + \int_0^t e^{As} B e^{A(t-s)}\, ds + \int_0^t \left(\int_0^s e^{Ar} B e^{A(s-r)}\, dr \right) B e^{A(t-s)}\, ds + \dots \quad (1)$$

which is a sum of recursive functions. This series, was first explored in quantum field theory (Dyson, 1949) and is called a series of time-ordered products (TOP), or sometimes a path-ordered exponential. In stochastic processes, Mjolsness & Yosiphon (2006) called it a time-ordered product expansion and used it to guide a sampling algorithm. We will employ the expansion to derive our deterministic method, Tree of Time-Ordered Products (TTOP).

2.1. TOP computation tree

We make two assumptions: Q can be split into $Q = A+B$, where ve^{At} is relatively simple to compute, and B is broken into J manageable terms $B = \sum_{j=1}^{J} B_j$. We will show how to realize these assumptions in the following sections. We use the TOP expansion of Equation 1 and apply the distributive property of matrix multiplication to move the sum outside the integral:

$$ve^{(A+\sum_{j=1}^{J} B_j)t} = ve^{At} + \sum_{j=1}^{J} \int_0^t ve^{As} B_j e^{A(t-s)}\, ds + \sum_{j=1}^{J} \sum_{j'=1}^{J} \int_0^t \left(\int_0^s ve^{Ar} B_j e^{A(s-r)}\, dr \right) B_{j'} e^{A(t-s)}\, ds + \dots \quad (2)$$

Let $F^l(t)$ represent the lth term of the expansion. Then the equation can be rewritten as

$$ve^{Qt} = \sum_{l=0}^{\infty} F^l(t) \quad (3)$$

where

$$F^0(t) = ve^{At}$$
$$F^l(t) = \sum_{j=1}^{J} \int_0^t F^{(l-1)}(s) B_j e^{-As}\, ds\; e^{At}. \qquad (4)$$

By construction, the first term, ve^{At}, is simple to solve (as will be explained in Section 2.2).

2.1.1. Integral expansion

We calculate each integral with the following expansion in which we treat a polynomial portion exactly and use adaptive quadrature to estimate the non-polynomial portion. Let $g(t)$ be a general function of time, g_0 be a constant, and $q(t)$ be a piece-wise polynomial. Further denote $\bar{q}(t) = \int_0^t q(s)\,ds$ and $q_{[r_1,r_2]}(t) = I[r_1 \le t < r_2]q(t)$ (both also piece-wise polynomials), where $I[\cdot]$ is the indicator function. Then we can write an integral of the form $h(t;q,g,g_0) = \int_0^t q(s)(g(s)-g_0)ds$ as

$$\begin{aligned} h(t;q,g,g_0) &= \bar{q}(t)(g(s_0)-g_0) \\ &\quad + \int_0^t q_{[a,s_0]}(s)\,(g(s)-g(s_0))\;ds \\ &\quad + \int_0^t q_{[s_0,b]}(s)\,(g(s)-g(s_0))\;ds \\ &= \bar{q}(t)(g(s_0)-g_0) \\ &\quad + h(t;q_{[a,s_0]},g,g(s_0)) + h(t;q_{[s_0,b]},g,g(s_0)) \qquad (5) \end{aligned}$$

for any chosen g_0, approximating g as the constant $g(s_0)$ and adding 2 correction terms (of the same form) for subparts of the interval. Here, $[a,b]$ is the support range of q, and $s_0 = \frac{a+b}{2}$. For convenience, we divide this range into two: $[a, \frac{a+b}{2}]$ and $[\frac{a+b}{2}, b]$. We use two subdivision in the following sections as well, but generalization to more than two sub-intervals is straightforward.

Recursive expansion Equation 5 will generate infinitely many terms in the form of $\bar{q}(t)(g(s_0)-g_0)$:

$$h(t;q,g,g_0) = \sum_{k=1}^{\infty} q_k(t)u_k \qquad (6)$$

where $q_k(t)$ is $\bar{q}(t)$ for a particular q, and u_k is $(g(s_0)-g_0)$ for a particular g and g_0.

2.1.2. Complete computation tree

Assume level l of Equation 3 can be expressed as

$$F^l(t) = \sum_{k=1}^{\infty} q_k^l(t)u_k^l e^{At} \qquad (7)$$

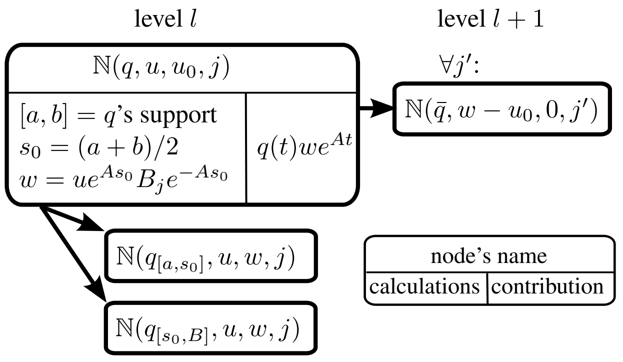


Figure 1. General form of the expansion.

(as is certainly true for $l=0$: $q_1^0(t)=1$, $u_1^0 = v$). We then show how to construct level $l+1$ similarly:

$$\begin{aligned} F^{l+1}(t) &= \sum_{j=1}^{J}\int_0^t F^l(s)B_je^{-As}\,ds\,e^{At} \\ &= \sum_{j=1}^{J}\int_0^t \sum_{k'=1}^{\infty} q_{k'}^l(s)u_{k'}^l e^{As}B_je^{-As}\,ds\,e^{At} \\ &= \sum_{k'=1}^{\infty}\sum_{j=1}^{J}\int_0^t q_{k'}^l(s)u_{k'}^l e^{As}B_je^{-As}\,ds\,e^{At} \\ &= \sum_{k'=1}^{\infty}\sum_{j=1}^{J} h(t;q_{k'}^l, u_{k'}^l e^{At}B_je^{-At},0)\,e^{At} \\ &= \sum_{k'=1}^{\infty}\sum_{j=1}^{J}\sum_{k=1}^{\infty} q_{k',k,j}^l(t)u_{k',k,j}^l e^{At}. \qquad (8) \end{aligned}$$

In last two lines, we have replaced the integral with the expansion of Equation 6. In particular, the integration of interest is $h(t;q_{k'}^l, u_{k'}^l e^{At}B_je^{-At},0)$. We let the set $\{q_k,u_k\}_k$ generated for this h be denoted as $\left\{q_{k',k,j}^l, u_{k',k,j}^l\right\}_k$.

In Equation 8, k' represents a node at level l in the computation tree. So, for every k', we generate J nodes in level $l+1$, each of which are the roots of trees for expansion of corresponding integrals. The result is an expansion for F^{l+1} of the same form as Equation 7.

We denote a term in this expansion as a compute node $\mathbb{N}(q,u,u_0,j)$. It and its descendants in the same level l represent $h(t;q,ue^{At}B_je^{-At},u_0)$. Its children in level $l+1$ represent new terms in F^{l+1}. Node $\mathbb{N}(q,u,u_0,j)$ contributes the term

$$q(t)we^{At}, \text{ where } w = ue^{As_0}B_je^{-As_0}, \qquad (9)$$

to the total sum. Its two children on the same level are $\mathbb{N}(q_{[a,s_0]},u,w,j)$ and $\mathbb{N}(q_{[s_0,b]},u,w,j)$ where $s_0 = \frac{a+b}{2}$.

Algorithm 1 TTOP Filter

```
Initialize priority queue $PQ$ with $\mathbb{N}(1, v, 0, 0)$
while $PQ$ not empty and compute time left do
  Let $\mathbb{N}(q, u, u_0, j) \leftarrow$ Pop($PQ$)
  Set $(a, b)$ be the support range of $q$
  Set $s_0 = \frac{a+b}{2}$, and $w = ue^{As_0}B_je^{-As_0}$
  Add $q(t)we^{At}$ to the running sum (see Equation 9)
  {Next line can be generalized to more than 2 splits}
  Add $\mathbb{N}(q_{[a,s_0]}, u, w, j)$ and $\mathbb{N}(q_{[s_0,b]}, u, w, j)$ to $PQ$
  for $j' = 1$ to $J$ do
    for $i = 1$ to $m$ do
      Add $\mathbb{N}(\bar{q}, d_i(w, u_0), 0, j')$ to $PQ$
```

On the next level, it generates a node $\mathbb{N}(\bar{q}, w - u_0, 0, j')$, for all $1 \leq j' \leq J$. Figure 1 shows this expansion.

For reasons that will be clear in the next section, we cannot form $w - u_0$ (even though we can form each individually). Thus, node $\mathbb{N}(\bar{q}, w - u_0, 0, j')$ could be described by two nodes: $\mathbb{N}(\bar{q}, w, 0, j')$ and $\mathbb{N}(\bar{q}, -u_0, 0, j')$. However, the second node will essentially "undo" calculations done elsewhere. We address this in the next section.

These recursively generated nodes represent an infinite tree whose sum is ve^{Qt}. Nodes have a single value, plus "same-level" children who represent finer approximations of the integral, and "next-level" children who represent one component of F^l for the next level l. The sum of all nodes at level l describes a system that evolves according to A (instead of Q), but for which at l time points (positioned anywhere), the correction $B = Q - A$ is applied. If $A = 0$, then each level is one term in the Taylor expansion of Q. The traditional $\frac{1}{i!}$ coefficients in such an expansion are captured by our piece-wise polynomial integrations.

If we explore the tree such that every node will be visited in the limit of infinite computational time, then we can add each node's evaluation at time t to a running sum and compute ve^{Qt}. Algorithm 1 outlines this method using a priority queue to concentrate computation on portions of the tree with large contributions.

2.2. Structured TOP calculations

For the scenarios of interest, the vector v and matrix Q are too large to explicitly represent because the state space consists of one state for every assignment to a set of m variables, X_1 through X_m. We will assume that v is an independent distribution (although we can extend this work to dependencies in v, but this eases the exposition). In Kronecker algebra this means $v = \bigotimes_{i=1}^m v_i$, where each v_i is a small row-vector of the marginal distribution over X_i.

We now show how to keep each quantity in the computation tree representable exactly as a similarly factored distribution: a Kronecker product with one term for each variable. If $A = \bigoplus_{i=1}^m A_i$, then $e^{At} = \bigotimes_{i=1}^m e^{A_it}$. We assume that each A_i (which is only over the state space of X_i) is small enough so that calculation of e^{A_it} can be performed efficiently. If we also require that each $B_j = \bigotimes_{i=1}^m B_{j,i}$, then all vectors and matrices to be multiplied in the computation tree are such Kronecker products. In particular, at node $\mathbb{N}(q, u, u_0, j)$, we need to compute w (Equation 9) for its contribution to the sum. Because u, B_j, and e^{As_0} are all Kronecker products and $(A\otimes B)(C\otimes D) = (AC)\otimes(BD)$, all of the matrix products can be performed efficiently by just operating on the subspaces over each variable independently. Thus w (and by extension the node's contribution to the sum) is a completely factored vector represented as a Kronecker product (that is, a distribution in which all variables are independent).

The generation of new nodes does not require any other operations, except for manipulation of one-dimensional piece-wise polynomials. Thus, our answer is a weighted sum of independent distributions (Kronecker products). It is not representable as a Kronecker product because it is a full distribution. However, any expectation of this distribution can be computed by summing up the contribution of each of these independent terms.

As we mentioned earlier, $\mathbb{N}(\bar{q}, w - u_0, 0, j')$ has the expression $w - u_0$ which is not a Kronecker product (despite that both w and u_0 are). To handle this, we note that

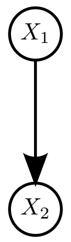
$$\bigotimes_{i=1}^m x_i - \bigotimes_{i=1}^m y_i = \sum_{i'=1}^m d_{i'}(x, y) \tag{10}$$

where

$$d_{i'}(x, y) = \left(\bigotimes_{i<i'} y_i\right) \otimes (x_{i'} - y_{i'}) \otimes \left(\bigotimes_{i>i'} x_i\right) \tag{11}$$

Because of how we select B_j (see below), $x_{i'} - y_{i'}$ is only non-zero for a few i' (the family of variable j). Furthermore, this allows us to split up the nodes by how much the deviation $(w - u_0)$ contributes by variable, which concentrates computation on those variables whose approximations are most difficult. Thus, we use this decomposition and divide the node $\mathbb{N}(\bar{q}, w - u_0, 0, j')$ (see Figure 1) into nodes $\mathbb{N}(\bar{q}, d_{i'}(w, u_0), 0, j')$ for all i' for which $w_{i'} \neq u_{0i'}$.

The necessary form for $Q = \bigoplus_{i=1}^m A_i + \sum_{j=1}^J \bigotimes_{i=1}^m B_{j,i}$ is always possible, although it might result in one B_j for each element of Q (which would in general be exponential in the number of state variables). Binary decision diagrams are often used to encode Q. In stochastic logic applications a disjunctive partitioning leads very naturally to this structure (Burch et al., 1991; Ciardo & Yu, 2005). However, they are encoded in a form where $A = 0$. Techniques similar to those we describe next can be applied to



$$Q_1 = \begin{bmatrix} -1 & 1 \\ 3 & -3 \end{bmatrix} \qquad A_1 = \begin{bmatrix} -1 & 1 \\ 3 & -3 \end{bmatrix}$$

$$Q_{2|0} = \begin{bmatrix} -2 & 2 \\ 4 & -4 \end{bmatrix} \qquad B_1 = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}}_{\Delta_{1,1}} \otimes \underbrace{\begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix}}_{Q_{2|1}-A_2=B_{(2|1),2}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} -2 & 2 \\ 3 & -3 \end{bmatrix}$$

$$Q_{2|1} = \begin{bmatrix} -4 & 4 \\ 3 & -3 \end{bmatrix} \qquad B_2 = \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}}_{\Delta_{2,2}} \otimes \underbrace{\begin{bmatrix} -2 & 2 \\ 0 & 0 \end{bmatrix}}_{Q_{2|2}-A_2=B_{(2|2),2}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q = \begin{bmatrix} -3 & 2 & 1 & 0 \\ 4 & -5 & 0 & 1 \\ 3 & 0 & -7 & 4 \\ 0 & 3 & 3 & -6 \end{bmatrix} \qquad A = A_1 \oplus A_2 = \begin{bmatrix} -3 & 2 & 1 & 0 \\ 3 & -4 & 0 & 1 \\ 3 & 0 & -5 & 2 \\ 0 & 3 & 3 & -6 \end{bmatrix} \qquad B = B_1 + B_2$$

Figure 2. Two-node CTBN and decomposition. The large matrices (4-by-4) are implicit. Q is broken into an independent A and two correction matrices, B_1 and B_2.

pull intensity from the B_j matrices into A, but we will focus on a different representation.

2.3. Continuous time Bayesian networks

A continuous time Bayesian network (CTBN) (Nodelman et al., 2002) is a graphical model which provides a structured representation of a CTMP. The initial distribution is described by a Bayesian network which we assume has no edges (but this work can be extended to dependencies in the initial distribution). The transition model is specified as a directed and possibly cyclic graph, in which the nodes in the model represent variables of the Markov process, and the dynamics of each node depend on the state of its parents in the graph. Each node X_i, for $i = 1, \ldots, m$, has a set of parents U_i. The rate matrix Q is factored into conditional rate matrices, $Q_{i|u_i}$ for every assignment of u_i to U_i. Each conditional intensity matrix gives the rates at which variable X_i transitions at instants when $U_i = u_i$. No two variables can transition at *exactly* the same instant so any element in the global Q matrix describing a change of multiple variables is 0.

The global Q matrix for a CTBN can be represented with Kronecker algebra. Let $R_{i|u_i} = \bigotimes_{i'} R_{(i|u_i),i'}$ be a Kronecker product of one matrix for each variable where

$$R_{(i|u_i),i'} = \begin{cases} Q_{i|u_i} & \text{if } i' = i \\ \Delta_{k,k} & \text{if } i' \in \text{Pa}(X_i) \text{ \& } k \text{ is val. of } i' \text{ in } u_i \\ I & \text{otherwise} \ . \end{cases} \tag{12}$$

where $\Delta_{k,k}$ is a matrix of all zeros except a single one at location k, k. In this way, $R_{i|u_i}$ distributes the rates in $Q_{i|u_i}$ to the proper locations in Q. The full Q for the CTBN is therefore

$$Q = \sum_{i=1}^{M} \sum_{u_i} \left(\bigotimes_{i'=1}^{M} R_{(i|u_i),i'} \right) \ . \tag{13}$$

This corresponds to our TOP representation of Q where A is 0 and there is one B_j for each variable and instantiation of its parents. We can pull intensity into the A matrix by defining

$$B_{(i|u_i),i'} = \begin{cases} Q_{i|u_i} - A_i & \text{if } i' = i \\ \Delta_{k,k} & \text{if } i' \in \text{Pa}(X_i) \text{ \& } k \text{ is val. of } i' \text{ in } u_i \\ I & \text{otherwise} \ . \end{cases} \tag{14}$$

Then

$$Q = \bigoplus_i A_i + \sum_{i=1}^{M} \sum_{u_i} \left(\bigotimes_{i'=1}^{M} B_{(i|u_i),i'} \right) \ . \tag{15}$$

The algebra is long, but straight-forward. It holds because A_i is constant with respect to u_i and the sum over u_i represents each possible instantiation exactly once. The result is that A_i represents an independent, approximate process for X_i. A is the joint process of each of these independent approximations. The differences between the independent process and the CTBN are given by one $B_{i|u_i}$ for each variable and its parents' instantiation. Note that $\Delta_{(i|u_i),i'} = I$ if i' is not a parent of i. Thus, most of the components of any B_j are the identity and computing $ue^{As}Be^{-As}$ for these components is trivial (they are the same as u). Thus, the calculations for $\mathbb{N}(q, u, u_0, j)$ are local to the variable j and its parents.

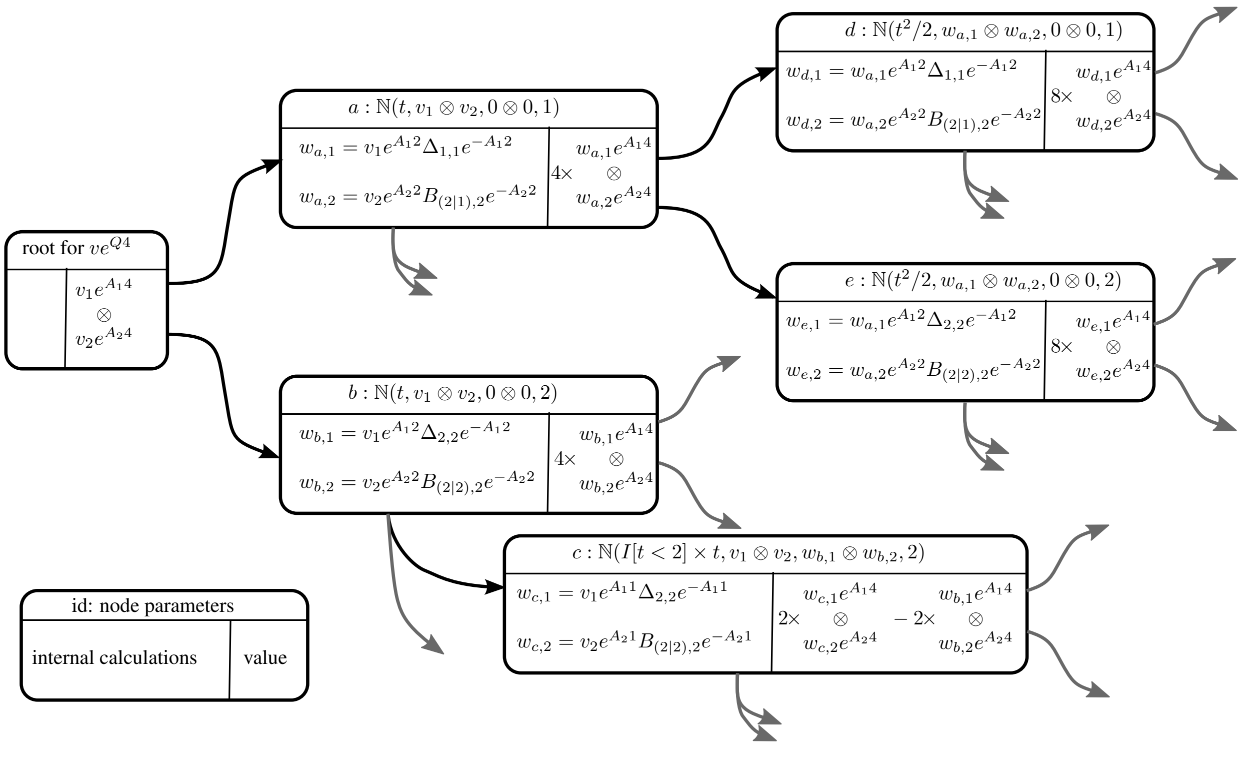

root for ve^{Q4}
$v_1e^{A_14}$ $\otimes$ $v_2e^{A_24}$
$a : \mathbb{N}(t, v_1 \otimes v_2, 0 \otimes 0, 1)$
$w_{a,1} = v_1e^{A_12}\Delta_{1,1}e^{-A_12}$
$w_{a,2} = v_2e^{A_22}B_{(2|1),2}e^{-A_22}$
$4\times$ $w_{a,1}e^{A_14} \otimes w_{a,2}e^{A_24}$
$b : \mathbb{N}(t, v_1 \otimes v_2, 0 \otimes 0, 2)$
$w_{b,1} = v_1e^{A_12}\Delta_{2,2}e^{-A_12}$
$w_{b,2} = v_2e^{A_22}B_{(2|2),2}e^{-A_22}$
$4\times$ $w_{b,1}e^{A_14} \otimes w_{b,2}e^{A_24}$
$d : \mathbb{N}(t^2/2, w_{a,1} \otimes w_{a,2}, 0 \otimes 0, 1)$
$w_{d,1} = w_{a,1}e^{A_12}\Delta_{1,1}e^{-A_12}$
$w_{d,2} = w_{a,2}e^{A_22}B_{(2|1),2}e^{-A_22}$
$8\times$ $w_{d,1}e^{A_14} \otimes w_{d,2}e^{A_24}$
$e : \mathbb{N}(t^2/2, w_{a,1} \otimes w_{a,2}, 0 \otimes 0, 2)$
$w_{e,1} = w_{a,1}e^{A_12}\Delta_{2,2}e^{-A_12}$
$w_{e,2} = w_{a,2}e^{A_22}B_{(2|2),2}e^{-A_22}$
$8\times$ $w_{e,1}e^{A_14} \otimes w_{e,2}e^{A_24}$
$c : \mathbb{N}(I[t < 2] \times t, v_1 \otimes v_2, w_{b,1} \otimes w_{b,2}, 2)$
$w_{c,1} = v_1e^{A_11}\Delta_{2,2}e^{-A_11}$
$w_{c,2} = v_2e^{A_21}B_{(2|2),2}e^{-A_21}$
$2\times$ $w_{c,1}e^{A_14} \otimes w_{c,2}e^{A_24}$ $-2\times$ $w_{b,1}e^{A_14} \otimes w_{b,2}e^{A_24}$
id: node parameters
internal calculations
value

Figure 3. Portion of computation tree for example in Figure 2 for $t = 4$. Each box is one node in the computation tree (see Figure 1). Nodes a and b are the children of the root. Nodes d and e are examples of their children at the next level ($l = 2$). Node c is an example of a refinement (of node b). Equation 11 dictates how the children of c for $l = 2$ are computed because $u_0 \neq 0$ for this node.

2.4. CTBN example

Figure 2 shows a simple 2-variable CTBN and one possible decomposition into A and B. Here the local A_i matrices are chosen to be the minimal rates. Because X_1 has no parents, A_1 is exact and there are no B terms. X_2 generates 2 B terms. If we let $v = v_1 \otimes v_2$ (that is, v_1 and v_2 are the independent marginals of X_1 and X_2), then Figure 3 shows a small portion of the computation tree.

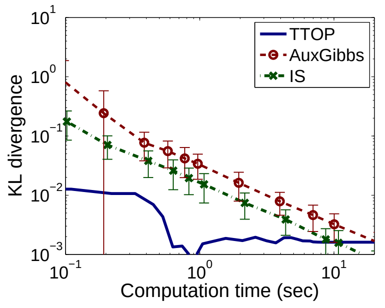
This method essentially computes the effect of A exactly and incorporates the effects of B as a Taylor expansion, adding increasing numbers of terms. Note that in Figure 2, A_1 and A_2 are proper intensity matrices (their rows sum to 0). This means that $B_{2|0}$ and $B_{2|1}$ have negative diagonal elements. This results in a computation that corresponds to a Taylor expansion with alternating signs. This can cause computational problems. One alternative is to arrange for B to have no negative elements. For instance $A_2 = \begin{bmatrix} -4 & 2 \\ 3 & -4 \end{bmatrix}$, resulting in $B_{2|0} = \begin{bmatrix} 2 & 0 \\ 1 & 0 \end{bmatrix}$ and $B_{2|1} = \begin{bmatrix} 0 & 2 \\ 0 & 1 \end{bmatrix}$. The disadvantage is that $v_2e^{A_2t}$ sums to less than 1. The non-root nodes add probability to the answer (instead of moving it within the answer).

Finally, for a CTBN, multiplication by a B_j is particularly simple. The j value indexes a variable (X_i) and an instantiation to its parents (u_i). To multiply a factored vector by B_j, multiply the X_i component by $Q_{i|u_i} - A_i$. For each component associated with a parent, zero out all elements of the vector except for the one consistent with u_i. Vectors for non-parent nodes are unchanged.

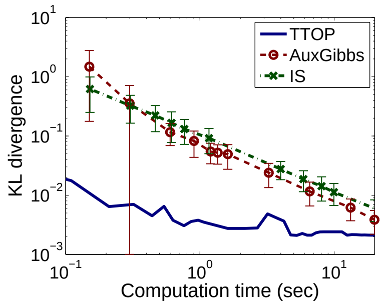
2.5. Smoothing and computational considerations

The discussion so far has focused on filtering. We would also like to perform smoothing. We will limit ourselves to the case in which the initial distribution at time 0, v, is known and there is evidence at a later time T for which the vector v_T represents the probability of the evidence for each possible state of the system. We assume that both vectors factor as previously discussed.

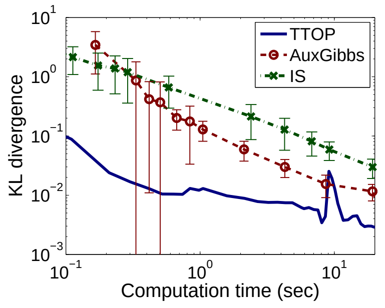
The goal is to compute (an expectation of) the distribution at time t conditioned on the evidence at time T. This consists of computing the Hadamard (point-wise) product of ve^{Qt} and $v_Te^{Q^\top(T-t)}$ (and then normalizing the result). First, consider computing each exponential separately. Let the result of the "forward" direction be $\sum_j \alpha_j$ where the forward calculation's jth node had contribution $\alpha_j = \bigotimes_{i=1}^m \alpha_{j,i}$. Let the "backward" direction be similarly



(a) $m = 9$

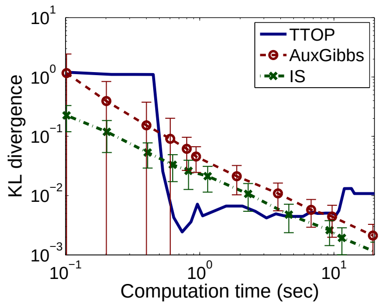


(b) $m = 15$

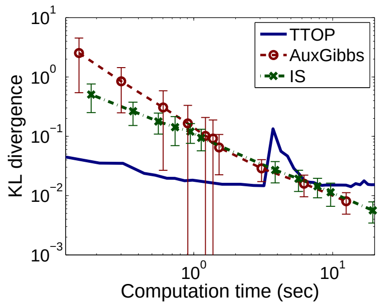


(c) $m = 21$

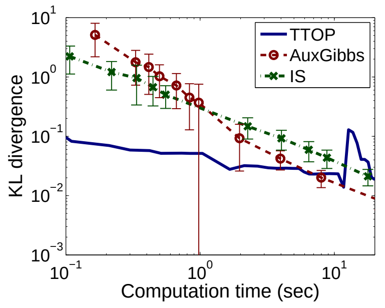
Figure 4. Computation time versus KL divergence for the toroid networks when $\tau = 2$, $\beta = 0.5$.



(a) $m = 9$



(b) $m = 15$



(c) $m = 21$

Figure 5. Computation time versus KL divergence for the toroid networks when $\tau = 2$, $\beta = 1$.

represented as $\sum_k \beta_k$ with $\beta_k = \bigotimes_{i=1}^{m} \beta_{k,i}$. It can easily be shown that

$$\sum_{j=1}^{N_a} \alpha_j \odot \sum_{k=1}^{N_b} \beta_k = \sum_{j=1}^{N_a} \sum_{k=1}^{N_b} \left(\bigotimes_{i=1}^{m} \alpha_{j,i} \right) \odot \left(\bigotimes_{i=1}^{m} \beta_{k,i} \right)$$
$$= \sum_{j=1}^{N_a} \sum_{k=1}^{N_b} \bigotimes_{i=1}^{m} (\alpha_{j,i} \odot \beta_{k,i}) \qquad (16)$$

Thus, we must consider every pair of nodes, one from the forward expansion and one from the backward expansion. For each pair, the components of the factored representation are point-wise multiplied together to obtain the pair's contribution to the answer.

We want to include these terms judiciously to best use a finite computational budget. We do this by keeping a frontier set of pairs of computation nodes, one from the forward tree and one from the backward tree. If we have explored (added to the smoothing result) $\alpha_j \odot \beta_k$, we add to our frontier set α_j paired with all of β_k's children and α_j's children paired with β_k. We use a closed list to ensure no pair is considered twice.

The question then is how to prioritize members of the frontier. We need to have an estimate of the total contribution of this pair and all subsequent pairs in the graph. We select the sum of this node's contribution to the query value (absolution value of the change) and the product of the maximum value in each node.

3. Experiments

We implemented our method, TTOP (Tree of Time-Ordered Products), as part of the CTBN-RLE code base (Shelton et al., 2010), and it will be included in the next version. We evaluated our method on a synthetic network of Ising model dynamics. The Ising model is a well-known interaction model with applications in many fields including statistical mechanics, genetics, and neuroscience (Zhou & Schmidler, 2009). The experimental results focus on inference accuracy given a known network. The Ising model was chosen so that we could compute the true answer in a reasonable time and scale the problem size.

Using this model, we generated a directed toroid network structure with cycles following (El-Hay et al., 2010). Nodes follow their parents' states according to a coupling strength parameter (β). A rate parameter (τ) determines how fast nodes toggle between states. We scale the number of nodes in the network but limit it to 21 to be able to compare the results to exact inference. We use three networks of respectively $m = 9$, 15, and 21 binary variables. We could not include networks with more than 21 binary variables because we cannot do exact exponentiation on a matrix of size bigger than $2^{21} \times 2^{21}$ in a reasonable time. We scale the network by adding rows of nodes. For these networks we fix the τ parameter and vary β. Nodes can take on values -1 and $+1$.

We compare TTOP to two efficient anytime algorithms: auxiliary Gibbs sampling (AuxGibbs) (Rao & Teh, 2011) and importance sampling (IS) (Fan et al., 2010). We also compared to a mean field variational approach (MF) (Cohn et al., 2009); however, error for MF is above the error range of other methods for all computation times. For this reason, we omit the MF results in the plots. We analyze the error in marginals computed by each method relative to exact inference. We focus on the computation time because in our experiments memory was not an issue and whole computation tree occupied only a few GBs.

For TTOP, we set the number of splits for the quadrature (see Equation 5) to 10 because it produces a good computation time versus error performance. We vary the computation time budget to observe the trade-off between computation time and the error.

For AuxGibbs, we vary the sample size between 50 and 5000, and set the burn-in period to be 10% of this value. For IS, the sample size varies between 500 to 50000. We ran the experiments 100 times for each test for both sampling methods. The computation time shown in the plots is the average of these runs for a given number of samples. The error is the sum of the KL-divergences of all the marginals from their true values.

Our experiments focus on smoothing. The networks start from a deterministic state, for $m = 9$: at $t = 0$ variables 1–5 are $+1$ and 6–9 are -1. At $t = 1$, variables 1–3 have switched to -1, 4–5 remain $+1$, and 6–9 have switched to $+1$. For $m = 15$ and 21 we use a similar pattern of evidence for comparison reasons. For $m = 15$, the variables 1–5, 7–8, 10–11 start at $+1$ and the remaining variables start at -1. The variables 1–3 switch to -1, while 4–5,7–8, and 10–11 stay at $+1$, and 6, 9 and 12–15 switch to $+1$. The evidence for $m = 21$ also follows the same pattern scaled to 21 nodes.

The nodes are not observed between $t = 0$ and $t = 1$. We query the marginal distributions of nodes at $t = 0.5$. Figures 4 and 5 show computation time versus sum of KL-divergence of marginals. We focus on the first 20 seconds of computation time because usually a few seconds are enough for our inference tasks. The lines in the plots continue their trend and cross at some point except for Figure 4-b. A KL-divergence sum of 10^{-2} is generally accurate for these networks.

Figure 4 shows the results for $\tau = 2$, $\beta = 0.5$. For most of these experiments, TTOP performs better than sampling methods. When the coupling strength of the network is increased to $\beta = 1$ as shown in Figure 5, TTOP has more variations in the error as the computation time increases but still has better performance overall. The occasional peaks in the error happen because sometimes a part of the computation tree is expanded and added to the sum, without the part that balances it since the time budget expired. This can be seen as more computation time is given to the algorithm, the errors decrease with the addition of the balancing part.

As the number of nodes in the network increase, our method keeps the computation time versus error advantage. Additionally, the gap between our method and others increases with the network size. Especially when $\beta = 1$, it performs better for the larger networks. While we cannot perform exact inference for larger networks, we expect these trends would continue as the problem size scales.

TTOP is also much better for short computation times, because it solves $e^{(At)}$ directly by integration while the sampling methods can generate only a few samples. Although the derivatives are smaller for the TTOP lines, this could potentially be fixed with better node prioritization. The best node prioritization would be one that looked at the contribution of the whole subtree rooted at a node rather than only the contribution of that node. Our heuristic is good for the first few levels of the tree, but it does not do as well as we go deeper in the tree.

The fluctuations in the error of TTOP are expected. The error from a single run of sampling would fluctuate as well. The plotted results of our method are from a single run compared to the averaged results of sampling methods which are 100 runs.

4. Conclusion

We have demonstrated an anytime algorithm for structured CTMP filtering and smoothing. Unlike prior work, it is deterministic, which can be of benefit when used inside learning methods. In the experiments, it has better computation time versus error performance than prior anytime convergent methods, especially for loosely coupled systems. Also as network size increases and coupling strength stays the same, our method's advantage increases as well.

Acknowledgments

This work was funded by The Laura P. and Leland K. Whittier Virtual PICU at Children's Hospital Los Angeles (awards UCR-12101024 and 8220-SGNZZ0777-00) and DARPA (award FA8750-14-2-0010).

References

Baier, Christel, Haverkort, Boudewijn, Hermanns, Holger, and Katoen, Joost-Pieter. Model checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, June 2003.

Burch, Jerry R., Clarke, Edmund M., and Long, David E. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, pp. 49–58, August 1991.

Ciardo, Gianfranco and Yu, Andy Jinqing. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Proceedings of Correct Hardware Design and Verification Methods*, pp. 146–161, 2005.

Cohn, Ido, El-Hay, Tal, Kupferman, Raz, and Friedman, Nir. Mean field variational approximation for continuous-time Bayesian networks. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*, 2009.

Dyson, F. J. The radiation theories of Tomonaga, Schwinger, and Feynman. *Physical Review*, 75(3):486–502, 1949.

El-Hay, Tal, Cohn, Ido, Friedman, Nir, and Kupferman, Raz. Continuous-time belief propagation. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 343–350, Haifa, Israel, June 2010.

Fan, Yu and Shelton, Christian R. Learning continuous-time social network dynamics. In *Proceedings of the Twenty-Fifth International Conference on Uncertainty in Artificial Intelligence*, 2009.

Fan, Yu, Xu, Jing, and Shelton, Christian R. Importance sampling for continuous time Bayesian networks. *Journal of Machine Learning Research*, 11(Aug):2115–2140, 2010.

Fernandes, Paulo, Plateau, Brigitte, and Stewart, William J. Efficient descriptor-vector multiplication in stochastic automata networks. *Journal of the ACM*, 45(3):381–414, 1998.

Gunawardana, Asela, Meek, Christopher, and Xu, Puyang. A model for temporal dependencies in event streams. In *Advances in Neural Information Processing Systems*, volume 24, 2012.

Mjolsness, Eric and Yosiphon, Guy. Stochastic process semantics for dynamical grammars. *Annals of Mathematics and Artificial Intelligence*, 47(3–4):329–395, August 2006.

Moler, Cleve and Loan, Charles Van. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

Najfeld, Igor and Havel, Timothy F. Derivatives of the matrix exponential and their computation. *Advances in Applied Mathematics*, 16:321–375, 1995.

Ng, Brenda, Pfeffer, Avi, and Dearden, Richard. Continuous time particle filtering. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1360–1365, 2005.

Nodelman, Uri, Shelton, Christian R., and Koller, Daphne. Continuous time Bayesian networks. In *Proceedings of the Eighteenth International Conference on Uncertainty in Artificial Intelligence*, pp. 378–387, 2002.

Rao, Vinayak and Teh, Yee Whye. Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In *Proceedings of the Twenty-Seventh International Conference on Uncertainty in Artificial Intelligence*, 2011.

Shelton, Christian R., Fan, Yu, Lam, William, Lee, Joon, and Xu, Jing. Continuous time Bayesian network reasoning and learning engine. *Journal of Machine Learning Research*, 11(Mar):1137–1140, 2010.

Xu, Jing and Shelton, Christian R. Intrusion detection using continuous time Bayesian networks. *Journal of Artificial Intelligence Research*, 39:745–774, 2010.

Zhou, Xiang and Schmidler, Scott C. Bayesian parameter estimation in Ising and Potts models: A comparative study with applications to protein modeling. Technical report, Duke University, 2009.

Appendix C

Faster Cover Trees

Appeared as

Mike Izbicki and Christian R. Shelton. Faster Cover Trees. In *Proceedings of the Thirty-Second International Conference on Machine Learning (ICML)*, 2015.

Faster Cover Trees

Mike Izbicki MIZBI001@UCR.EDU
Christian Shelton CSHELTON@CS.UCR.EDU
University of California Riverside, 900 University Ave, Riverside, CA 92521

Abstract

The cover tree data structure speeds up exact nearest neighbor queries over arbitrary metric spaces (Beygelzimer et al., 2006). This paper makes cover trees even faster. In particular, we provide

1. A simpler definition of the cover tree that reduces the number of nodes from $O(n)$ to exactly n,
2. An additional invariant that makes queries faster in practice,
3. Algorithms for constructing and querying the tree in parallel on multiprocessor systems, and
4. A more cache efficient memory layout.

On standard benchmark datasets, we reduce the number of distance computations by 10–50%. On a large-scale bioinformatics dataset, we reduce the number of distance computations by 71%. On a large-scale image dataset, our parallel algorithm with 16 cores reduces tree construction time from 3.5 hours to 12 minutes.

1. Introduction

Data structures for fast nearest neighbor queries are most often used to speed up the k-nearest neighbor classification algorithm. But many other learning tasks also require neighbor searches, and cover trees can speed up these tasks as well: localized support vector machines (Segata & Blanzieri, 2010), dimensionality reduction (Lisitsyn et al., 2013), and reinforcement learning (Tziortziotis et al., 2014). Making cover trees faster—the main contribution of this paper—also makes these other tasks faster.

Given a space of points $\mathscr{X}$, a dataset $X \subseteq \mathscr{X}$, a data point $p \in \mathscr{X}$, and a distance function $d : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$, the nearest neighbor of p in X is defined as

$$p_{nn} = \underset{q \in X - \{p\}}{\arg\min}\, d(p,q)$$

The naive method for computing p_{nn} involves a linear scan of all the data points and takes time $\theta(n)$, but many data structures have been created to speed up this process. The kd-tree (Friedman et al., 1977) is probably the most famous. It is simple and effective in practice, but it can only be used on Euclidean spaces. We must turn to other data structures when given an arbitrary metric space. The simplest and oldest of these structures is the *ball tree* (Omohundro, 1989). Although attractive for its simplicity, it provides only the trivial runtime guarantee that queries will take time $O(n)$. Subsequent research focused on providing stronger guarantees, producing more complicated data structures like the *metric skip list* (Karger & Ruhl, 2002) and the *navigating net* (Krauthgamer & Lee, 2004). Because these data structures were complex and had large constant factors, they were mostly of theoretical interest. The cover tree (Beygelzimer et al., 2006) simplified navigating nets while maintaining good run time guarantees. Further research has strengthened the theoretical runtime bounds provided by the cover tree (Ram et al., 2010). Our contributions make the cover tree faster in practice.

The cover tree as originally introduced is *simpler* than related data structures, but it is not *simple*. The original explanation required an implicit tree with an infinite number of nodes; but a smaller, explicit tree with $O(n)$ nodes actually gets implemented. We refer to this presentation as the *original cover tree*. In the remainder of this paper we introduce the *simplified cover tree* and the *nearest ancestor cover tree*; parallel construction and querying algorithms; and a cache-efficient layout suitable for all three cover trees. We conclude with experiments showing: on standard benchmark datasets we outperform both the original implementation (Beygelzimer et al., 2006) and MLPack's implementation (Curtin et al., 2013a); on a large bioinformatics dataset, our nearest ancestor tree uses 71% fewer distance comparisons than the original cover tree; and on a large image data set, our parallelization algorithm reduces tree construction time from 3.5 *hours* to 12 *minutes*.

Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37. Copyright 2015 by the author(s).

2. Simplified cover trees

Definition 1. *Our* simplified cover tree *is any tree where: (*a*) each node p in the tree contains a single data point (also denoted by p); and (*b*) the following three invariants are maintained.*

1. The leveling invariant. *Every node p has an associated integer* $\texttt{level}(p)$. *For each child q of p*

$$\texttt{level}(q) = \texttt{level}(p) - 1 \ .$$

2. The covering invariant. *For every node p, define the function* $\texttt{covdist}(p) = 2^{\texttt{level}(p)}$. *For each child q of p*

$$d(p,q) \leq \texttt{covdist}(p) \ .$$

3. The separating invariant. *For every node p, define the function* $\texttt{sepdist}(p) = 2^{\texttt{level}(p)-1}$. *For all distinct children q_1 and q_2 of p*

$$d(q_1,q_2) > \texttt{sepdist}(p)$$

Throughout this paper, we will use the functions $\texttt{children}(p)$ and $\texttt{descendants}(p)$ to refer to the set of nodes that are children or descendants of p respectively. We also define the function $\texttt{maxdist}$ as

$$\texttt{maxdist}(p) = \operatorname*{arg\,max}_{q \in \texttt{descendants}(p)} d(p,q)$$

In words, this is the greatest distance from p to any of its descendants. This value is upper bounded by $2^{\texttt{level}(p)+1}$, and its exact value can be cached within the data structure.[1]

In order to query a cover tree (any variant), we use the generic "space tree" framework developed by Curtin *et al.* (2013). This framework provides fast algorithms for finding the k-nearest neighbors, points within a specified distance, kernel density estimates, and minimum spanning trees. Each of these algorithms has two variants: a *single tree* algorithm for querying one point at a time, and a *dual tree* algorithm for querying many points at once. Our faster cover tree algorithms speed up all of these queries; but for simplicity, in this paper we focus only on the single tree nearest neighbor query. The pseudocode is shown in Algorithm 1.

Analysis of the cover tree's runtime properties is done using the data-dependent *doubling constant c*: the minimum value c such that every ball in the dataset can be covered by c balls of half the radius. We state without proof two facts: (*a*) any node in the cover tree can have at most $O(c^4)$ children; (*b*) the depth of any node in the cover tree is at most $O(c^2 \log n)$. These were proven for the original cover tree (Beygelzimer et al., 2006) and the proofs for our simplified cover tree are essentially the same. We can use these two facts to show that the runtime of Algorithm 1 is $O(c^6 \log n)$ for both the original and simplified cover tree.

Algorithm 1 Find nearest neighbor

```
function findNearestNeighbor(cover tree $p$, query point $x$, nearest neighbor so far $y$)
1: if $d(p,x) < d(y,x)$ then
2:     $y \leftarrow p$
3: for each child $q$ of $p$ sorted by distance to $x$ do
4:     if $d(y,x) > d(y,q) - \texttt{maxdist}(q)$ then
5:         $y \leftarrow \texttt{findNearestNeighbor}(q,x,y)$
6: return $y$
```

Algorithm 2 Simplified cover tree insertion

```
function insert(cover tree $p$, data point $x$)
1: if $d(p,x) > \texttt{covdist}(p)$ then
2:     while $d(p,x) > 2\texttt{covdist}(p)$ do
3:         Remove any leaf $q$ from $p$
4:         $p' \leftarrow$ tree with root $q$ and $p$ as only child
5:         $p \leftarrow p'$
6:     return tree with $x$ as root and $p$ as only child
7: return $\texttt{insert\_}(p,x)$
function insert_(cover tree $p$, data point $x$)
prerequisites: $d(p,x) \leq \texttt{covdist}(p)$
1: for $q \in \texttt{children}(p)$ do
2:     if $d(q,x) \leq \texttt{covdist}(q)$ then
3:         $q' \leftarrow \texttt{insert\_}(q,x)$
4:         $p' \leftarrow p$ with child $q$ replaced with $q'$
5:         return $p'$
6: return $p$ with $x$ added as a child
```

Algorithm 2 shows how to insert into the simplified cover tree. It is divided into two cases. In the first case, we cannot insert our data point x into the tree without violating the covering invariant. So we raise the level of the tree p by taking any leaf node and using that as the new root. Because $\texttt{maxdist}(p) \leq 2\texttt{covdist}(p)$, we are guaranteed that $d(p,x) \leq \texttt{covdist}(x)$, and so we do not violate the covering constraint. In the second case, the $\texttt{insert\_}$ function recursively descends the tree. On each function call, we search through $\texttt{children}(p)$ to find a node we can insert into without violating the covering invariant. If we find such a node, we recurse; otherwise, we know we can add x to $\texttt{children}(p)$ without violating the separating invariant. In all cases, exactly one node is added per data point, so the resulting tree will have exactly n nodes. Since every

[1]As in the original cover tree, practical performance is improved on most datasets by redefining $\texttt{covdist}(p) = 1.3^{\texttt{level}(p)}$ and $\texttt{sepdist}(p) = 1.3^{\texttt{level}(p)-1}$. All of our experiments use this modified definition.

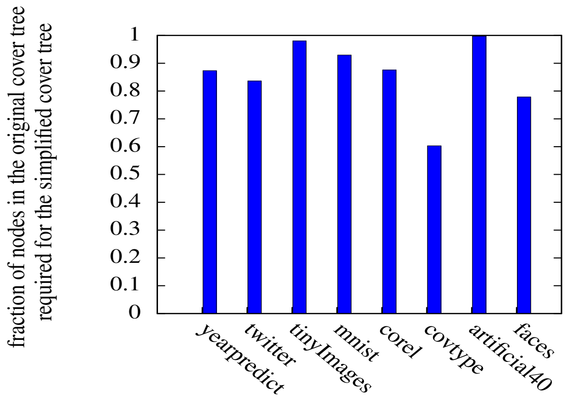


Figure 1. Fraction of nodes required for the simplified cover tree. Fewer nodes means less overhead from traversing the tree and fewer distance comparisons. See Table 1 for information on the datasets.

node can have at most $O(c^4)$ children and the depth of the tree is bounded by $O(c^2 \log n)$, the runtime is $O(c^6 \log n)$.

The simplified cover tree has the same runtime bounds as the original cover tree, but it has an improved constant factor, as it needs only n nodes. Fewer nodes reduces both the overhead from traversing the data structure and the number of required distance comparisons. The original cover tree's *nesting invariant* dictated these extra nodes. In our definitions, the nesting invariant states that for every node p, if p has any children, then p also has itself as a child (this child need not satisfy the leveling invariant). The nesting invariant comes from the presentation of the original cover tree as an infinite data structure, but it does not play a key role in the cover tree's runtime analysis. Therefore, we can discard it and maintain the runtime guarantees.

Figure 1 shows the reduction in nodes by using the simplified cover tree on benchmark datasets taken from the MLPack test suite (Curtin et al., 2013a). Section 6 contains more details on these datasets, and Figure 3 in the same section shows how this reduced node count translates into improved query performance.

3. Nearest ancestor cover trees

In this section, we exploit a similarity between simplified cover trees and binary search trees (BSTs). Insertion into both trees follows the same procedure: Perform a depth first search to find the right location to insert the point. In particular, there is no rebalancing after the insertion. Many alternatives to plain BSTs produce better query times by introducing new invariants. These invariants force the insertion algorithm to rebalance the tree during the insertion step. Our definition of the simplified cover tree makes adding similar invariants easy. We now introduce one possible invariant.

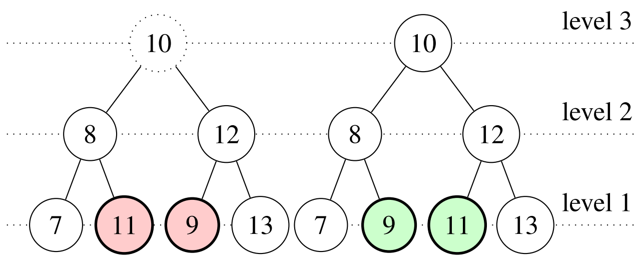


Figure 2. Using the metric $d(a,b) = |a-b|$, both trees are valid simplified cover trees; but only the right tree is a valid nearest ancestor cover tree. Moving the 9 and 11 nodes reduces the value of `maxdist` for their ancestor nodes. This causes pruning to happen more often during nearest neighbor queries.

Definition 2. *A* nearest ancestor cover tree *is a simplified cover tree where every point p has the nearest ancestor invariant: If q_1 is an ancestor of p and q_2 is a sibling of q_1, then*

$$d(p,q_1) \leq d(p,q_2)$$

In other words, the nearest ancestor cover tree ensures that for every data point, each of its ancestors is the "best possible" ancestor for it at that level. Figure 2 shows a motivating one dimensional example.

Algorithm 3 shows how to insert a point into a nearest ancestor cover tree. It uses the same `insert` function as 2, but the helper function `insert_` is slightly modified in two ways (shown with an underline). First, we sort `children`(p) according to their distance from the data point x. This sorting ensures that our newly inserted point will satisfy the nearest ancestor invariant. But this new point x may cause other points to violate the nearest ancestor invariant. In particular, if x has a sibling q; q has a descendent r; and $d(r,x) < d(r,q)$; then r now violates the nearest ancestor invariant. Our second step is to call `rebalance`, which finds all these violating data points and moves them underneath x.

Most of the work of the `rebalance` function happens in the helper `rebalance_`. `rebalance_` returns a valid nearest ancestor cover tree and a set of points that still need to be inserted; `rebalance` just inserts those extra points. `rebalance_` takes two nearest ancestor cover trees p and q (where p is an ancestor of q) and a point x. Its goal is to "extract" all points from q that would violate the nearest ancestor invariant if x became a sibling of p. It returns three values: a modified version of q, a set of points that cannot remain in any point along the path from p to q called the *moveset*, and a set of points that need to be reinserted somewhere along the path from p to q called the *stayset*. There are two cases. In the first case, the data point at node q must move. We then filter the descendants of q into the *moveset* or *stayset* as appropriate and return `null` for our modified q. In the second case, the data point at node q must stay. We recursively apply `rebalance_` to each of q's children; we use the results to update the corresponding child, the *moveset* and the *stayset* variables. Finally, we try to reinsert any nodes in *stayset*. If `rebalance_` was called with q a child of p, then the return value of *stayset* will be empty; any children that could not be directly reinserted must be in the *moveset*.

The `rebalance_` function loops over all $O(c^4)$ children of a node, and the maximum depth of the recursion is $O(c^2 \log n)$. Therefore the overall runtime is $O(c^6 \log n)$. It may be called up to $O(c^4)$ times within `rebalance`, so the body of the for loop on line 12 executes $O(c^{10} \log n)$ times. Unfortunately, we have no bound on the size of *moveset*, except to note that it is usually small in practice. On the datasets in our experiments (see Table 1), the value is usually zero or at worst in the single digits. Figure 3(a) shows that nearest ancestor cover tree construction is not that much slower in practice, and Figure 3(b) shows that this slowdown is overshadowed by the resulting speedup in nearest neighbor queries.

4. Parallel cover tree

In this section we discuss parallelism on shared-memory, multiprocessor machines. Querying in parallel is easy. Since the results of neighbor queries for a data point do not depend on other data points, we can: divide the points among the processors; then each processor traverses the tree independently. More difficult is constructing the tree in parallel. Our strategy is to split the data, create one cover tree on each processor, then merge these trees together. Previous work on parallelizing cover trees applied only to the GPU (Kumar & Ramareddy, 2010). Our approach is suitable for any shared-memory multiprocessor machine. We give a detailed description for merging simplified cover trees and discuss at a high level how to extend this procedure to nearest ancestor cover trees.

Algorithm 4 shows the merging procedure. The `merge` function's main purpose is to satisfy the prerequisites for `mergeHelper`, which has two phases. First, we find all the subtrees of q that can be inserted directly into p without violating any invariants, and we insert them. Second, we insert the remaining nodes from q into p directly via the `insert` function.

The `mergeHelper` function returns a partially merged tree and a set of nodes called the *leftovers* that still need to be inserted into the tree. The first phase uses the for loop starting on line 3 to categorize the children of q into three disjoint sets. The *uncovered* set contains all of q's children that would violate the covering invariant if inserted into p. The *sepcov* set contains all of q's children that would not

Algorithm 3 Nearest ancestor cover tree insertion

```
function insert_(cover tree $p$, data point $x$)
 1: for $q \in$ children(p) $\underline{\text{sorted by distance to } x}$ do
 2:     if $d(q,x) \leq$ covdist(q) then
 3:         $q' \leftarrow$ insert_($q,x$)
 4:         $p' \leftarrow p$ with child $q$ replaced with $q'$
 5:         return $p'$
 6: return $\underline{\texttt{rebalance}(p, x)}$

function rebalance(cover trees $p$, data point $x$)
prerequisites: $x$ can be added as a child of $p$ without violating the covering or separating invariants
 1: create tree $x'$ with root node $x$ at level level(p) $- 1$ $x'$ contains no other points
 2: $p' \leftarrow p$
 3: for $q \in$ children(p) do
 4:     $(q', moveset, stayset) \leftarrow$ rebalance_($p,q,x$)
 5:     $p' \leftarrow p'$ with child $q$ replaced with $q'$
 6:     for $r \in moveset$ do
 7:         $x' \leftarrow$ insert($x', r$)
 8: return $p'$ with $x'$ added as a child

function rebalance_(cover trees $p$ and $q$, point $x$)
prerequisites: $p$ is an ancestor of $q$
 1: if $d(p,q) > d(q,x)$ then
 2:     $moveset, stayset \leftarrow \emptyset$
 3:     for $r \in$ descendants(q) do
 4:         if $d(r,p) > d(r,x)$ then
 5:             $moveset \leftarrow moveset \cup \{r\}$
 6:         else
 7:             $stayset \leftarrow stayset \cup \{r\}$
 8:     return (null, $moveset, stayset$)
 9: else
10:     $moveset', stayset' \leftarrow \emptyset$
11:     $q' \leftarrow q$
12:     for $r \in$ children(q) do
13:         $(r', moveset, stayset) \leftarrow$ rebalance_($p,r,x$)
14:         $moveset' \leftarrow moveset \cup moveset'$
15:         $stayset' \leftarrow stayset \cup stayset'$
16:         if $r' =$ null then
17:             $q' \leftarrow q$ with the subtree $r$ removed
18:         else
19:             $q' \leftarrow q$ with the subtree $r$ replaced by $r'$
20:     for $r \in stayset'$ do
21:         if $d(r,q)' \leq$ covdist(q)$'$ then
22:             $q' \leftarrow$ insert($q', r$)
23:             $stayset' \leftarrow stayset' - \{r\}$
24:     return ($q', moveset', stayset'$)
```

violate the separating or covering invariants when inserted into p. Both of these sets are unused in the second phase of `mergeHelper`. Every child of q that is not inserted into the *uncovered* or *sepcov* sets gets merged with a suitable node in `children`(p). This is done by recursively calling the `mergeHelper` function. Any points that could not be inserted into the results of `mergeHelper` get added to the *leftovers* set.

In the second phase of `mergeHelper`, we insert as many nodes as possible into our merged tree p'. First update the children with the subtrees in *sepcov*. Then insert the root of q. We know that $d(p,q) \leq$ `covdist`(p), so this insertion is guaranteed not to change the level of p'. Finally, we loop over the elements in *leftovers* and insert them into p' only if it would not change the level of p'. Any elements of *leftovers* that cannot be inserted into p' get inserted into *leftovers*$'$ and returned. It is important to do this insertion of *leftovers* at the lowest level possible (rather than wait until the recursion ends and have the insertion performed in `merge`) to avoid unnecessary distance computations.

The `merge` function does not maintain the nearest ancestor invariant. A modified version of `merge` that calls the `rebalance` function appropriately could. But for space reasons, we do not provide this modified algorithm. In our experiments below, we use the provided `merge` function in Algorithm 4 to parallelize both simplified and nearest ancestor tree construction. In practice, this retains the benefits of the nearest ancestor cover tree because the nearest ancestor invariant is violated in only a few places.

Providing explicit bounds on the runtime of `merge` is difficult. But in practice it is fast. When parallelizing on two processors, approximately 1% of the distance calculations occur within `merge`. So this is not our bottleneck. Instead, the main bottleneck of parallelism is cache performance. On modern desktop computers, last level cache is shared between all cores on a CPU. Cover tree construction results in many cache misses, and this effect is exaggerated when the tree is constructed in parallel.

Algorithm 4 Merging cover trees

```
function merge(cover tree $p$, cover tree $q$)
 1: if level(q) $>$ level(p) then
 2:     swap $p$ and $q$
 3: while level(q) $<$ level(p) do
 4:     move a node from the leaf of $q$ to the root;
 5:     this raises the level of $q$ by 1
 6: $(p, leftovers) \leftarrow$ mergeHelper$(p,q)$
 7: for $r \in leftovers$ do
 8:     $p \leftarrow$ insert$(p,r)$
 9: return $p$
function mergeHelper(cover tree $p$, cover tree $q$)
prereqs: level(p) = level(q), $d(p,q) \leq$ covdist(p)
 1: $children' \leftarrow$ children(p)                          ▷ Phase 1
 2: $uncovered, sepcov, leftovers \leftarrow \emptyset$
 3: for $r \in$ children(q) do
 4:     if $d(p,r) <$ covdist(p) then
 5:         $foundmatch \leftarrow$ false
 6:         for $s \in children'$ do
 7:             if $d(s,r) \leq$ sepdist(p) then
 8:                 $(s', leftovers_s) \leftarrow$ mergeHelper$(s,r)$
 9:                 $children' \leftarrow children' \cup \{s'\} - \{s\}$
10:                 $leftovers \leftarrow leftovers \cup leftovers_s$
11:                 $foundmatch \leftarrow$ true
12:                 break from inner loop
13:         if not $foundmatch$ then
14:             $sepcov \leftarrow sepcov \cup \{r\}$
15:     else
16:         $uncovered \leftarrow uncovered \cup \{r\}$
17: $children' \leftarrow children' \cup sepcov$                  ▷ Phase 2
18: $p' \leftarrow$ tree rooted at $p$ with children(p')=$children'$
19: $p' \leftarrow$ insert$(p',q)$
20: $leftovers' \leftarrow \emptyset$
21: for $r \in leftovers$ do
22:     if $d(r,p)' \leq$ covdist(p)$'$ then
23:         $p' \leftarrow$ insert$(p',r)$
24:     else
25:         $leftovers' \leftarrow leftovers' \cup \{r\}$
26: return $(p', leftovers' \cup uncovered)$
```

5. Cache efficiency

One of the biggest sources of overhead in the cover tree is cache misses. Our last improvement is to make cover trees more cache efficient. A simple way to reduce cache misses for tree data structures is to use the *van Emde Boas* tree layout (Frigo et al., 1999). This layout arranges nodes in memory according to a depth first traversal of the tree. This arrangement creates a *cache oblivious* data structure. That is, the programmer does not need any special knowledge about the cache sizes to obtain optimal speedup—the van Embde Boas tree layout works efficiently on any cache architecture. This layout has been known for a long time in the data structures community, but it seems unused in machine learning libraries. Frigo (1999) provides a detailed tutorial.

Our implementation of the cache oblivious cover tree is *static*. That is, we first construct the cover tree, then we call a function `pack` that rearranges the tree in memory. This means we do not get the reduced cache misses while constructing the tree, but only while querying the tree. The `pack` function is essentially free to run because it requires only a single traversal through the dataset. Figure 4 shows that the van Emde Boas tree layout reduces cache misses by 5 to 20 percent. This results in a reduction of stalled CPU cycles by 2 to 15 percent.

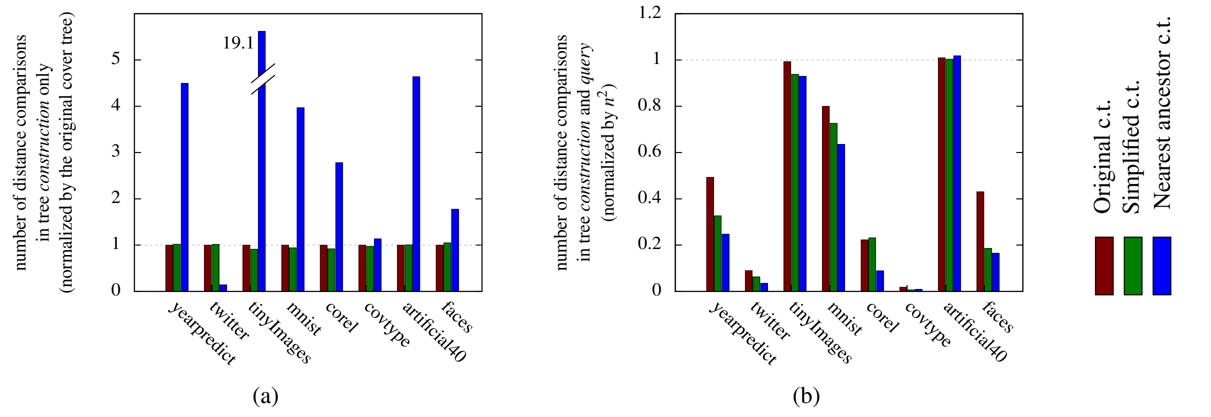


Figure 3. (a) Constructing a nearest ancestor query tree usually takes longer than the original cover tree and the simplified cover tree. (b) Construction *plus* querying is faster in the nearest ancestor cover tree. On most datasets, this faster query time more than offsets the increased construction cost, giving an overall speedup.

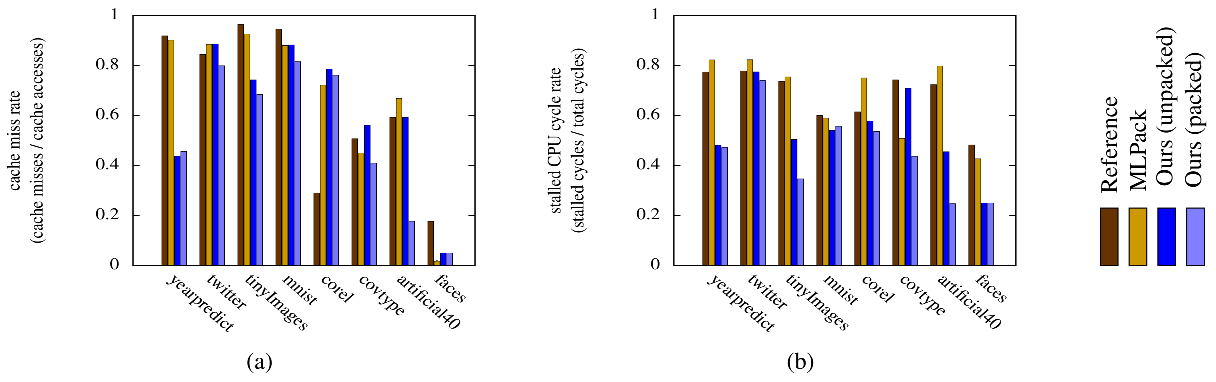


Figure 4. (a) Comparison of our packed nearest ancestor cover tree to our unpacked tree and other implementations, demonstrating better cache performance. (b) A stalled CPU cycle is when the CPU does no work because it must wait for a memory access. Reducing the number of cache misses results in fewer stalled cycles, and so faster run times. We used the Linux `perf stat` utility to measure the `cache-references`, `cache-misses`, `cycles`, and `stalled-cycles-frontend` hardware counters. `perf stat` uses a sampling strategy with negligible affect on program performance.

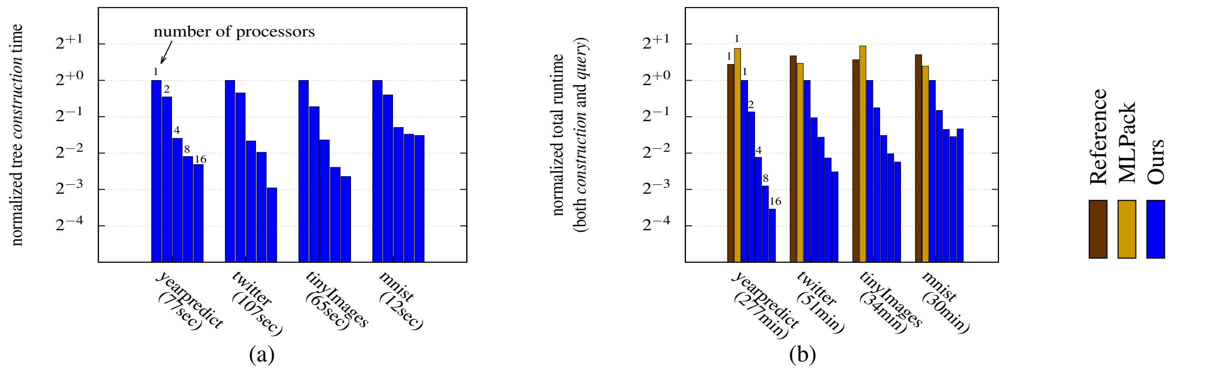


Figure 5. Run times on the "all nearest neighbor" procedure for only those datasets that take more than 5 minutes. (a) Tree construction. A single cover tree merge takes about 1% of the computation time; the main reason for the lack of perfect parallel speedup is the increased number of cache misses caused by inserting into multiple trees simultaneously. (b) Comparison on total performance to reference and MLPack implementations. Runtimes in both figures are divided by that of our single processor implementation (shown in parenthesis).

dataset	num data points	num dimensions
yearpredict	515345	90
twitter	583250	78
tinyImages	100000	384
mnist	70000	784
corel	68040	32
covtype	581012	55
artificial40	10000	40
faces	10304	20

Table 1. All MLPack benchmark datasets with at least 20 dimensions and 10000 points, arranged in descending order by runtime of all nearest neighbor search.

6. Experiments

We now validate our improvements empirically. Our first experiments use the Euclidean distance on a standard benchmark suite (described in Table 1). Our last experiments use non-Euclidean metrics on data from bioinformatics and computer vision. In each experiment, we use the "all nearest neighbors" experimental setup. That is, we first construct the cover trees on the dataset. Then, for each point in the dataset, we find its nearest neighbor. This is a standard technique for measuring the efficiency of nearest neighbor algorithms.

6.1. Tree-type comparison

Our first experiment compares the performance of the three types of cover trees: original, simplified, and nearest ancestor. We measure the number of distance comparisons required to build the tree on a dataset in Figure 3(a) and the number of distance comparisons required to find each data point's nearest neighbor in Figure 3(b) using Algorithm 1. Distance comparisons are a good proxy measure of runtime performance because the majority of the algorithm's runtime is spent computing distances, and it ignores the possible unwanted confounding variable of varying optimization efforts. As expected, the simplified tree typically outperforms the original tree, and the nearest ancestor tree typically outperforms the simplified tree. We reiterate that this reduced need for distance comparisons translates over to all other queries provided by the space tree framework (Curtin et al., 2013b).

6.2. Implementation comparison

We next compare our implementation against two good cover tree implementations currently in widespread use: the reference implementation used in the original paper (Beygelzimer et al., 2006) and MLPack's implementation (Curtin et al., 2013a). Both of these programs were written in C++ and compiled using `g++ 4.4.7` with full optimizations. Our implementation was written in Haskell and compiled with `ghc 7.8.4` also with full optimizations.[2] All tests were run on an Amazon Web Services `c3.8x-large` instance with 60 GB of RAM and 32 Intel Xeon E5-2680 CPU cores clocked at 2.80GHz. Half of those cores are hyperthreads, so for simplicity we only parallelize out to 16 cores.

Since the reference implementation and MLPack only come with the Euclidean distance built-in, we only use that metric when comparing the three implementations. Figure 4 shows the cache performance of all three libraries. Figure 5 shows the runtime of all three libraries. Our implementation's cache performance and parallelization speedup is shown on the nearest ancestor cover tree. Neither the original implementation nor MLPack support parallelization.

6.3. Graph kernels and protein function

An important problem in bioinformatics is to predict the function of a protein based on its 3d structure. State of the art solutions model the protein's 3d structure as a graph and use support vector machines (with a graph kernel) for prediction. Computing graph kernels is relatively expensive, however, so research has focused on making the graph kernel computation faster (Vishwanathan et al., 2010; Shervashidze et al., 2011). Such research makes graph kernels scale to *larger graphs*, but does not help in the case where there are *more graphs*. Our contribution is to use cover trees to reduce the number of required kernel computations, letting us scale to more graphs. The largest dataset in previous research contained about 1200 proteins. With our cover tree, we perform nearest neighbor queries on all one hundred thousand proteins currently registered in the Protein Data Bank (Berman et al., 2000).

We use the random walk graph kernel in our experiment. It performs well on protein classification and is conceptually simple. See Vishwanathan *et al.* (2010) for more details. A naive computation of this kernel takes time $O(v^6)$, where v is the number of vertices in the graph. Vishwanathan *et al.* present faster methods that take time only $O(v^3)$. While considerably faster, it is still a relatively expensive distance computation.

The Protein Data Bank (Berman et al., 2000) contains information on the 3d primary structure of approximately one hundred thousand proteins. To perform our experiment, we follow a procedure similar to that used by the PROTEIN dataset used in the experiments in Viswanathan *et al.*. This procedure constructs secondary structure graphs from the primary structures in the Protein Data Bank using the tool VLPG (Schäfer et al., 2012). The Protein Data Bank stores the 3d structure of the atoms in the protein in a PDB file.

[2]Our code can be downloaded at `http://github.com/mikeizbicki/hlearn#covertree`.

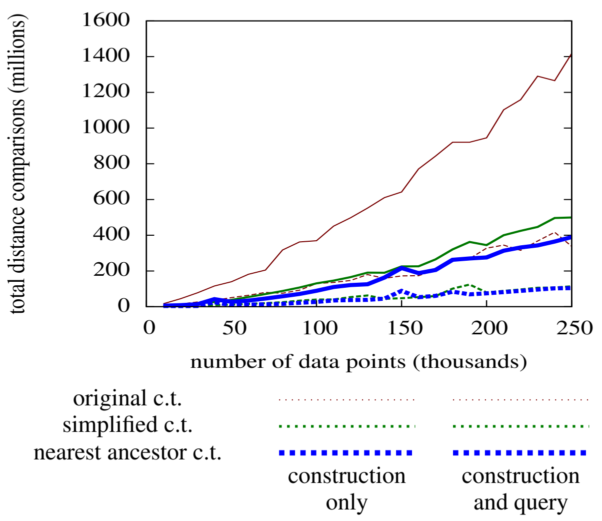


Figure 6. The effect on runtime as we increase the number of data points on the bionformatics data. The relationship is roughly linear, indicating protein graphs have a relatively low intrinsic dimensionality. As expected, the nearest ancestor cover tree performs the best.

From this PDB file, we calculate the protein's secondary structure using the DSSP tool (Joosten et al., 2011). Then, the tool VLPG (Schäfer et al., 2012) generates graphs from the resulting secondary structure. Some PDB files contain information for multiple graphs, and some do not contain enough information to construct a graph. In total, our dataset consists of 250,000 graphs, and a typical graph has between 5-120 nodes and 0.1-3 edges per node. Figure 6 shows the scaling behavior of all three cover trees on this dataset. On all of the data, the total construction and query cost are 29% that of the original cover tree.

6.4. Earth mover's distance

The Earth Mover's Distance (EMD) is a distance metric between histograms designed for image classification (Rubner et al., 1998). In our tests, we convert images into three dimensional histograms of the pixel values in LabCIE color space. LabCIE is a color space represents colors in three dimensions. It is similar to the more familiar RGB and CMYK color spaces, but the distances between colors more accurately match what humans perceive color distances to be. We construct the histogram such that each dimension has 8 equally spaced intervals, for a total of 512 bins. We then create a "signature" of the histogram by recording only the 20 largest of the 512 bins.

Previous research on speeding up EMD focused on computing EMD distances faster. The EMD takes a base distance as a parameter. For an arbitrary base distance, EMD requires $O(b^3 \log b)$ time where b is the size of the histogram signature. Faster algorithms exist for specific base metrics. For example, with an L_1 base metric the EMD can be computed in time $O(b^2)$ (Ling & Okada, 2007); and if the base metric is a so-called "thresholded metric," we can get an order of magnitude constant factor speed up (Pele & Werman, 2009). We specifically chose the LabCIE color space because there is no known faster EMD algorithm. It will stress-test our cover tree implementation.

number of cores	simplified tree construction time	simplified tree construction speedup	nearest ancestor tree construction time	nearest ancestor tree construction speedup
1	70.7 min	1.0	210.9 min	1.0
2	36.6 min	1.9	94.2 min	2.2
4	18.5 min	3.8	48.5 min	4.3
8	10.2 min	6.9	25.3 min	8.3
16	6.7 min	10.5	12.0 min	17.6

Table 2. Parallel cover tree construction using the earth movers distance. On this large dataset with an expensive metric, we see better parallel speedup than on the datasets with the cheaper L2 metric. The nearest ancestor cover tree gets super-linear parallel speedup because we are merging with Algorithm 4, which does not attempt to rebalance.

In this experiment, we use the Yahoo! Flickr Creative Commons dataset. The dataset contains 1.5 million images in its training set, and we construct simplified and nearest ancestor cover trees in parallel on this data. Construction times are shown in Table 2. Using the cheap L2 distance with smaller datasets, tree construction happens quickly and so parallelization is less important. But with an expensive distance on this larger dataset, parallel construction makes a big difference.

7. Conclusion

We've simplified the definition of the cover tree, and introduced the new nearest ancestor invariant that speeds up the cover tree in practice. It is possible that other invariants exist that will balance the tree better, providing even more speed improvements.

Acknowledgments

Shelton was supported by DARPA award FA8750-14-2-0010.

References

Berman, Helen M., Westbrook, John, Feng, Zukang, Gilliland, Gary, Bhat, T. N., Weissig, Helge, Shindyalov, Ilya N., and Bourne, Philip E. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

Beygelzimer, Alina, Kakade, Sham, and Langford, John. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 97–104, New York, NY, USA, 2006.

Curtin, Ryan R., Cline, James R., Slagle, Neil P., March, William B., Ram, P., Mehta, Nishant A., and Gray, Alexander G. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14: 801–805, 2013a.

Curtin, Ryan R, March, William B, Ram, Parikshit, Anderson, David V, Gray, Alexander G, and Isbell Jr, Charles L. Tree-independent dual-tree algorithms. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1435–1443, 2013b.

Friedman, Jerome H, Bentley, Jon Louis, and Finkel, Raphael Ari. An algorithm for finding best matches in logarithmic expected time. *Transactions on Mathematical Software*, 3(3):209–226, 1977.

Frigo, Matteo, Leiserson, Charles E, Prokop, Harald, and Ramachandran, Sridhar. Cache-oblivious algorithms. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pp. 285–297, 1999.

Joosten, Robbie P., te Beek, Tim A. H., Krieger, Elmar, Hekkelman, Maarten L., Hooft, Rob W. W., Schneider, Reinhard, Sander, Chris, and Vriend, Gert. A series of PDB related databases for everyday needs., January 2011.

Karger, David R. and Ruhl, Matthias. Finding nearest neighbors in growth-restricted metrics. In *In 34th Annual ACM Symposium on the Theory of Computing*, pp. 741–750, 2002.

Krauthgamer, Robert and Lee, James R. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 798–807, Philadelphia, PA, USA, 2004.

Kumar, R. Deepak and Ramareddy, K. Design and implementation of cover tree algorithm on cuda-compatible gpu. *International Journal of Computer Applications*, 3 (7):24–27, June 2010. Published By Foundation of Computer Science.

Ling, Haibin and Okada, Kazunori. An efficient earth mover's distance algorithm for robust histogram comparison. *PAMI*, 29:853, 2007.

Lisitsyn, Sergey, Widmer, Christian, and Garcia, Fernando J. Iglesias. Tapkee: An efficient dimension reduction library. *Journal of Machine Learning Research*, 14:2355–2359, 2013.

Omohundro, Stephen Malvern. Five balltree construction algorithms. Technical Report 89-063, International Computer Science Institute, December 1989.

Pele, Ofir and Werman, Michael. Fast and robust earth mover's distances. In *ICCV*, 2009.

Ram, Parikshit, Lee, Dongryeol, March, William, and Gray, Alexander G. Linear-time Algorithms for Pairwise Statistical Problems. In *Advances in Neural Information Processing Systems*, 2010.

Rubner, Yossi, Tomasi, Carlo, and Guibas, Leonidas J. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision*, pp. 59–66, 1998.

Schäfer, Tim, May, Patrick, and Koch, Ina. Computation and Visualization of Protein Topology Graphs Including Ligand Information. In *German Conference on Bioinformatics 2012*, volume 26, pp. 108–118, Dagstuhl, Germany, 2012.

Segata, Nicola and Blanzieri, Enrico. Fast and scalable local kernel machines. *Journal of Machine Learning Research*, 11:1883–1926, August 2010.

Shervashidze, Nino, Schweitzer, Pascal, van Leeuwen, Erik Jan, Mehlhorn, Kurt, and Borgwardt, Karsten M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, November 2011.

Tziortziotis, Nikolaos, Dimitrakakis, Christos, and Blekas, Konstantinos. Cover Tree Bayesian Reinforcement Learning. *Journal of Machine Learning Research*, 15, 2014.

Vishwanathan, S. V. N., Schraudolph, Nicol N., Kondor, Risi, and Borgwardt, Karsten M. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, August 2010.

Appendix D

Auxiliary Gibbs Sampling for Inference in Piecewise-Constant Conditional Intensity Models

Appeared as

Zhen Qin and Christian R. Shelton. Auxiliary Gibbs Sampling for Inference in Piecewise-Constant Conditional Intensity Models. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

Auxiliary Gibbs Sampling for Inference in Piecewise-Constant Conditional Intensity Models

Zhen Qin
University of California, Riverside
zqin001@cs.ucr.edu

Christian R. Shelton
University of California, Riverside
cshelton@cs.ucr.edu

Abstract

A piecewise-constant conditional intensity model (PCIM) is a non-Markovian model of temporal stochastic dependencies in continuous-time event streams. It allows efficient learning and forecasting given complete trajectories. However, no general inference algorithm has been developed for PCIMs. We propose an effective and efficient auxiliary Gibbs sampler for inference in PCIM, based on the idea of thinning for inhomogeneous Poisson processes. The sampler alternates between sampling a finite set of auxiliary virtual events with adaptive rates, and performing an efficient forward-backward pass at discrete times to generate samples. We show that our sampler can successfully perform inference tasks in both Markovian and non-Markovian models, and can be employed in Expectation-Maximization PCIM parameter estimation and structural learning with partially observed data.

1 Introduction

Modeling temporal dependencies in event streams has wide applications. For example, users' behaviors in online shopping and web searches, social network activities, and machines' responses in datacenter management can each be viewed as a stream of events over time. Models that can successfully learn the complex dependencies among events (both label and timing) allow targeted online advertising, automatic policy selection in datacenter management, user behavior modeling, or event prediction and dependency understanding in general.

[Gunawardana et al., 2011] proposed the piecewise-constant conditional intensity model (PCIM) which captures the dependencies among the types of events through a set of piecewise-constant conditional intensity functions. A PCIM is represented as a set of decision trees, which allow for efficient model selection. Forecasting via forward sampling is also simple by iteratively sampling next events based on the current history.

However, currently model selection and forecasting for PCIMs is only effective given complete data. When there are missing data, an inference method is needed to answer general queries or be employed in expectation-maximization (EM) algorithms for model selection and parameter learning. Currently, no inference algorithm has been proposed for PCIM that can condition on general evidence.

In this work, we propose the first general inference algorithm for PCIMs, based on the idea of *thinning* for inhomogeneous Poisson process [Lewis and Shedler, 1979]. This results in an auxiliary Gibbs sampler that alternates between sampling a finite set of virtual event times given the current trajectory, and then sampling a new trajectory given the set of evidences and event times (virtual and actual). Our method is convergent, does not involve approximations like fixed time-discretization, and the samples generated can answer any type of query. We propose an efficient state-vector representation to maintain only necessary information for diverging trajectories, reducing the exponentially increasing sampling complexity to linear in most cases. We show empirically our inference algorithm converges to the true distribution, permits effective query answering, and aids model selection with incomplete data for PCIM models with both Markovian and complex non-Markovian dynamics. We also show the connection between PCIMs and continuous-time Bayesian networks (CTBNs), and compare our method with an existing method on such models.

2 Previous Work

A dynamic Bayesian network (DBN) [Dean and Kanazawa, 1988] models temporal dependencies between variables in discrete time. For systems that evolve asynchronously without a global clock, it is often not clear how timestamps should be discretized. Health records, computer server logs, and social networks are examples of asynchronous event data streams. For such systems, too slow a sampling rate would poorly represent the data, while too fast a sampling rate makes learning and inference more costly.

Continuous-time models have drawn attention recently in applications ranging from social networks [Du et al., 2013, Saito et al., 2009, Linderman and Adams, 2014] to genetics [Cohn et al., 2009] to biochemical networks [Golightly and Wilkinson, 2011]. Continuous Time Bayesian Networks (CTBN) [Nodelman et al., 2002] are homogeneous *Markovian* models of the joint trajectories of discrete finite variables, analogous to DBNs. Non-Markovian continuous models allow the rate of an event to be a function of the process's history. Poisson Networks [Rajaram et al., 2005] constrain this function to depend only on the counts of the number of events during a finite time window. Poisson cascades [Simma and Jordan, 2010] define the rate function to be the sum of a kernel applied to each historic event, and requires the modeler to choose a parametric form for temporal dependencies.

A PCIM defines the intensity function as decision trees, with internal nodes' tests mapping time and history to leaves. Each leaf is associated with a constant rate. A PCIM is able to model non-Markovian temporal dependencies, and is an order of magnitude faster to learn than Poisson networks. Applications include modeling supercomputer event logs and forecasting future interests of web search users. While PCIMs have been extended in a number of ways [Parikh et al., 2012, Weiss and Page, 2013], there is no general inference algorithms.

Inference algorithms developed for continuous systems are mainly for Markovian models or specifically designed for a particular application. For CTBNs, there are variational approaches such as expectation propagation [El-Hay et al., 2010] and mean field [Cohn et al., 2009], which do not converge to the true value as computation time increases. Sampling based approaches include importance sampling [Fan et al., 2010] and Gibbs sampling [Rao and Teh, 2011, Rao and Teh, 2013] that converge to the true value. The latter is the current state-of-the-art method designed for general Markov Jump Processes (MJPs) and its extensions (including CTBNs). It uses the idea of uniformization [Grassmann, 1977] for Markov models, similar to thinning [Lewis and Shedler, 1979] for inhomogeneous Poisson processes. We note that our inference method generalizes theirs to non-Markovian models.

3 PCIM Background

Assume events are drawn from a finite label set L. An event then can be represented by a time-stamp t and a label l. An event sequence $x = \{(t_i, l_i)\}_{i=1}^n$, where $0 < t_1 < \ldots < t_n$. We use $h_i = \{(t_j, l_j) \mid (t_j, l_j) \in x, t_j < t_i)\}$ for the history of event i, when it is clear from context which x is meant. We define the ending time $t(y)$ of an event sequence y as the time of the last event in y, so that $t(h_i) = t_{i-1}$. A conditional intensity model (CIM) is a set of non-negative conditional intensity functions indexed by label $\{\lambda_l(t|x;\theta)\}_{l=1}^{|L|}$. The data likelihood is

$$p(x|\theta) = \prod_{l \in L} \prod_{i=1}^{n} \lambda_l(t_i|h_i;\theta)^{\mathbf{1}_l(l_i)} e^{-\Lambda_l(t_i|h_i;\theta)} \qquad (1)$$

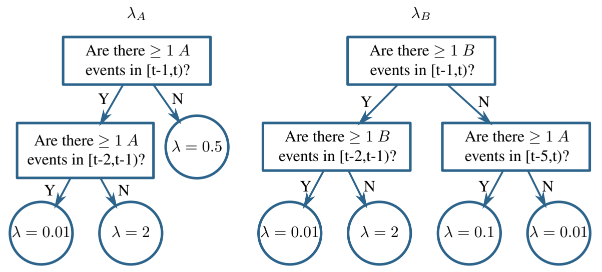
where $\Lambda_l(t|h;\theta) = \int_{t(h)}^{t} \lambda_l(\tau|h;\theta)d\tau$. The indicator function $\mathbf{1}_l(l')$ is one if $l' = l$ and zero otherwise. $\lambda_l(t|h;\theta)$ is the expected rate of event l at time t given history h and model parameters θ. Conditioning on the entire history causes the process to be non-Markovian. The modeling assumptions for a CIM are quite weak, as any distribution for x in which the timestamps are continuous random variables can be written in this form. Despite the weak assumptions, the per-label conditional factorization allows the modeling of label-specific dependence on past events.

A PCIM is a particular class of CIM that restricts $\lambda(h)$ to be piecewise constant (as a function of time) for any history, so the integral for Λ breaks down into a finite number of components and forward sampling becomes feasible. A PCIM represents the conditional intensity functions with decision trees. Each internal node in a tree is a binary test of the history, and each leaf contains an intensity. If the tests are piecewise-constant functions of time for any event history, the resulting function $\lambda(t|h)$ is piecewise-constant. Examples of admissible tests include

- Was the most recent event of label l?
- Is the time of the day between 6am and 9am?
- Did an event with label l happen at least n times between 5 seconds ago and 2 seconds ago?
- Were the last two events of the same label?

Note some tests are non-Markovian in that they require knowledge of more than just which event was most recent. See Fig. 1 for an example of a PCIM model.

The decision tree for label l maps the time and history to a



λ_A
Are there ≥ 1 A events in [t-1,t)?
Y
N
Are there ≥ 1 A events in [t-2,t-1)?
$\lambda = 0.5$
Y
N
$\lambda = 0.01$
$\lambda = 2$
λ_B
Are there ≥ 1 B events in [t-1,t)?
Y
N
Are there ≥ 1 B events in [t-2,t-1)?
Are there ≥ 1 A events in [t-5,t)?
Y
N
$\lambda = 0.01$
$\lambda = 2$
Y
N
$\lambda = 0.1$
$\lambda = 0.01$

Figure 1: Decision tree representing S and θ for events of labels A and B. Note the dependency among event labels (the rate of B depends on A). [Gunawardana et al., 2011]

leaf $s \in \Sigma_l$, where Σ_l is the set of leaves for l. The resulting data likelihood can be simplified:

$$p(x|S,\theta) = \prod_{l \in L} \prod_{s \in \Sigma_l} \lambda_{ls}^{c_{ls}(x)} e^{-\lambda_{ls} d_{ls}(x)}. \qquad (2)$$

S is the PCIM structure represented by the decision trees; the model parameters θ are rates at the leaves. $c_{ls}(x)$ is the number of times label l occurs in x and is mapped to leaf s. $d_{ls}(x)$ is the total duration when the event trajectory for l is mapped to s. c and d are the sufficient statistics for calculating data likelihood.

[Gunawardana et al., 2011] showed that given the structure S, by using a product of Gamma distributions as a conjugate prior for θ, the marginal likelihood of the data can be given in closed form, and thus parameter estimation can be done in closed form. Furthermore, imposing a structural prior allows a closed form Bayesian score to be used for greedy tree learning.

4 Auxiliary Gibbs Sampling for PCIM

In this section we introduce our new inference algorithm for PCIM, called ThinnedGibbs, based on the idea of *thinning* for inhomogeneous Poisson processes. We handle incomplete data in which there are intervals of time during which events for particular label(s) are not observed.

4.1 Why Inference in PCIM is Difficult

Filling in partially observed trajectories for PCIM is hard due to the complex dependencies between unobserved events and both past and future events. See Fig. 2 for an example. While the history (the event at t) says it is likely that there should be events in the unobserved area (with an expected rate of 2), future evidence (no events in R) is contradictory: If there were indeed events in the unobserved area, those events should stimulate events happening in R.

Such a phenomenon might suggest existing algorithms such as the forward-filtering-backward-sampling (FFBS) algorithm for discrete-time Markov chains. However, there are two subtleties here: First, we are dealing with non-Markovian models. Second, we are dealing with continuous-time systems, so the number of time steps over which to propagate is infinite.

4.2 Thinning

Thinning [Lewis and Shedler, 1979] can be used to turn a continuous-time process into a discrete-time one, without using a fixed time-slice granularity. We select a rate λ^* greater than any in the inhomogeneous Poisson process and sample from a *homogeneous* process with this rate. To get a sample from the original inhomogeneous process, an event at time t is thinned (dropped) with probability $1 - \frac{\lambda(t)}{\lambda^*}$.

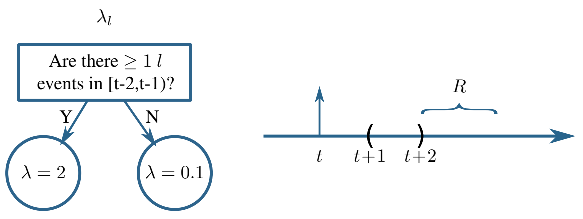


Figure 2: A simple PCIM with a partially observed trajectory. The vertical solid arrow indicates an evidence event. Areas between parentheses are unobserved. History alone indicates there should be events filled in, while the future (no events in R) provides contradictory evidence.

This process can also be reversed. If given the set of thinned event times (samples from the inhomogeneous process), extra events can be added to a sample from the original constant-rate process by sampling from a Poisson process with rate $\lambda^* - \lambda(t)$. The cycle can then repeat by thinning the new total set of times (ignoring how they were generated). At each cycle, the times (after thinning) are drawn from the original inhomogeneous process. It is this type of cycle we will employ in our sampler.

The difficulty is a PCIM is not an inhomogeneous Poisson process. Its intensity depends on the entire history of events, not just the current time. For thinning, this means that we cannot independently sample whether each event is to be thinned. Furthermore, we wish to sample from the posterior process, conditioned on evidence. All evidence (both past and future) affect the probability of a specific thinning configuration.

4.3 Overview of Our Method

To overcome both of these problems, we extend thinning to an auxiliary Gibbs sampler in the same way that [Rao and Teh, 2011, Rao and Teh, 2013] extended Markovian-model uniformization [Grassmann, 1977] (a specific example of thinning in a Markov process) to a Gibbs sampler. To do this we introduce auxiliary variables representing the events that were dropped. We call these events *virtual* events.

As a standard Gibbs sampler, our method cycles through each variable in turn. In our case, a variable corresponds to an event label. For event label l, let x_l be the sampled event sequence for this label. Let Y_l be all evidence (for l and other labels) and all (currently fixed) samples for other labels. Our goal is to sample from $p(x_l \mid Y_l)$.

Let v_l be the virtual events (the auxiliary variable) associated with l and $z_l = x_l \cup v_l$ (all event times, virtual and non-virtual). Our method first samples from $p(v_l \mid x_l, Y_l)$ and then samples from $p(x_l \mid z_l, Y_l)$. The first step adds virtual events given the non-virtual events are "correct." The second step treats all events as potential events and drops or keeps events. The dropped events are removed completely. The kept events, x_l, remain as the new sampled trajectory.

The proof of correctness follows analogously to that of [Rao and Teh, 2013] for Markovian systems. However, the details for sampling from $p(v_l \mid x_l, Y_l)$ and $p(x_l \mid z_l, Y_l)$ differ. We describe them next.

4.4 Sampling Auxiliary Virtual Events with Adaptive Rates

Sampling from $p(v_l \mid x_l, Y_l)$ amounts to adding just the virtual (dropped) events. As the full trajectory (x_l for all l) is known, the rate at any time step for a virtual event is independent of any other virtual events. Therefore, the process is an inhomogeneous Poisson process for which the rate at t is equal to $\lambda^* - \lambda_l(t|h)$ where h is fully determined by x_l and Y_l. Recall that $\lambda_l(t|h)$ is piecewise-constant in time, so sampling from such an inhomogeneous Poisson process is simple.

The auxiliary rate, λ^*, must be strictly greater than the maximum rate possible for irreducibility. We use an auxiliary rate of $\lambda^* = 2\max(\lambda(t|h))$ to sample virtual events in the unobserved intervals. This choice balances mixing time (better with higher λ^*) and computational complexity (better with lower λ^*).

A naïve way to pick λ^* is to find λ_{max}: the maximum rate in the leaves of PCIM, and use $2\lambda_{max}$. However, there could be unobserved time intervals with a possible maximum rate much smaller than λ_{max}. Using λ_{max} in those regions would generate too many virtual events, most of which will be dropped in the next step leading to computational inefficiency. We therefore use an adaptive strategy.

Our adaptive $\lambda^*(t|h)$ cannot depend on x_l (this would break the simplicity of sampling mentioned above). Therefore, we determine $\lambda^*(t|h)$ by passing (t, h) down the PCIM tree for λ_l. At each internal node, if the branch does not depend on x_l, we can directly take one branch. Otherwise, the test is related to the sampled events, and we take the maximum rate of taking both branches. This method results in $\lambda^*(t|h)$ as a piecewise-constant function of time (for the same reasons that $\lambda_l(t|h)$ is piecewise-constant).

Consider Fig. 3 as an example. When sampling event $l = A$ on the interval $[1, 5)$, we would not take the left branch at the root (no matter what events for A have been sampled), but must maximize over the other two leaves (as different x_l values would result in different leaves). This results in a $\lambda^* = 4$ over this interval, which is smaller than 6.

4.5 The Naïve FFBS Algorithm

Once these virtual events are added back in, we take z_l (the union of virtual events and "real" sampled events) as a sample from the Poisson process with rate λ^* and ignore which

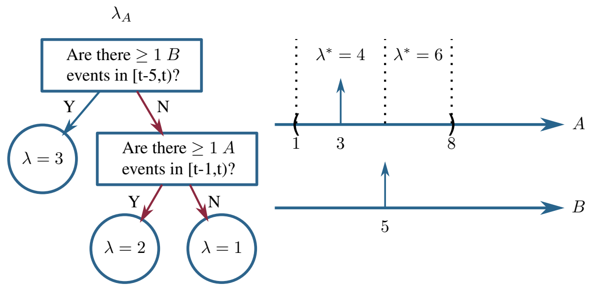


Figure 3: Adaptive auxiliary rate example. When sampling A, the branch to take at the root does not depend on unobserved events for A. If the test is related to the sampled event, we take the maximum rate from both branches. The red arrows indicate the branches to take between time $[1, 5]$, and $\lambda^* = 2 \times 2$ in that interval, instead of 6.

were originally virtual and which were originally "real." We then thin this set to get a sample from the conditional marginal over l.

The restriction to consider events only at times in z_l transforms the continuous-time problem into a discrete one. Given z_l with m possible event times $(z_{l,1}, z_{l,2}, \ldots, z_{l,m})$, let $b = \{b_i\}_{i=1}^m$ be a set of binary variables, one per event, where $b_i = 1$ if event i is included in x_l (otherwise $b_i = 0$ and the event is not included in x_l). Thus sampling b is equivalent to sampling x_l (z_l is known) as it specifies which events in z_l are in x_l. Let $Y_l^{i:j}$ be the portion of Y between times $z_{l,i}$ and $z_{l,j}$, and $b^{i:j} = \{b_k | i \le k \le j\}$ We wish to sample b (and thereby x_l) from $p(b \mid Y) \propto$

$$\left(\prod_i p(Y_l^{i-1:i}, b_i \mid b^{1:i-1}, Y_l^{1:i-1})\right) p(Y_l^{m:\infty} \mid b) \quad (3)$$

where the final $Y_l^{m:\infty}$ signifies all of the evidence after the last virtual event time $z_{l,m}$ and can be handled similarly to the other terms.

The most straight-forward method for such sampling considers each possible assignment to b (of which there are 2^m). For each interval, we multiply terms from Eq. 3 of the form $p(Y_l^{i-1:i}, b_i \mid b^{1:i-1}, Y_l^{1:i-1}) =$

$$p(Y_l^{i-1:i} \mid b^{1:i-1}, Y_l^{1:i-1}) p(b_i \mid b^{1:i-1}, Y_l^{1:i}) \quad (4)$$

where the first term is the likelihood of the trajectory interval from $z_{l,i-1}$ to $z_{l,i}$ and the second term is the probability of the event being thinned, given the past history. The first can be computed by tallying the sufficient statistics (counts and durations) and applying Eq. 2. Note that these sufficient statistics take into account $b^{1:i-1}$ which specifies events for l during the unobserved region(s), and the likelihood must also be calculated for labels $l' \neq l$ for which $\lambda_{l'}(t|h)$ depends on events from l. The second term is equal to $\frac{\lambda_l(t|h)}{\lambda^*(t)}$ if $b_i = 1$ (and $1 - \frac{\lambda_l(t|h)}{\lambda^*(t)}$ if $b_i = 0$). The numerator's dependence on the full history similarly dictates a dependence on $b^{1:i-1}$.

This might be formulated as a naïve FFBS algorithm: To generate one sample, we propagate possible trajectories forward in time, multiplying in Eq. 4 at each inter-event interval to account for the evidence. Every time we see a virtual event, each possible trajectory diverges into two (depending on whether the virtual event is to be thinned or not). By the end, we have all 2^m possible trajectories, each with its probability (Eq. 3). We sample one trajectory as the output, in proportion of the calculated likelihoods. As we explicitly keep all possible trajectories, the sampled trajectory immediately tells us which virtual events are kept, so no actual backward pass is needed.

4.6 An Efficient State-Vector Representation

The naïve FFBS algorithm is clearly not practical, as the number of possible trajectories grows exponentially with the number of auxiliary virtual events (m). We propose a more efficient state-vector representation to only keep the necessary information for each possible trajectory. The idea takes advantage of the structure of the PCIM and leads to state merges, similar to what happens in FFBS for hidden Markov models (HMMs).

The terms in Eq. 4 depend on $b^{1:i-1}$ only through the tests in the internal nodes of the PCIM trees. Therefore, we do not have to keep track of all of $b^{1:i-1}$ to calculate these likelihoods, but only the current state of such tests that depend on events with label l. For example, a test that asks "Is the last event of label l?" only needs to maintain a bit as the indicator. The test "Are there more than 3 events of label q in the last 5 seconds?" for $q \neq l$ has no state, as $b^{1:i-1}$ does not affect its choice. By contrast, a test such as "Is the last event of label q?" does depend on b, even if $q \neq l$.

As we propagate forward, we merge $b^{1:i}$ sequences that result in the same set of states for all internal tests inside the PCIM. See Fig. 4 as a simple example. Though there are 8 possible trajectories, they merge to only 2 states that we can sample from. Similar to FFBS for HMM, we need to maintain the transition probabilities in the forward pass and use them in a backward sampling pass to recover the full trajectory, but such information is also linear.

Note that this conversion to a Markov system for sampling is *not* possible in the original continuous-time system. Thinning has allows it by randomly selecting a few discrete time points, thereby restricting the possible state space to be finite.

The state space depends on the actual tests in the PCIM model. See Tbl. 1 for the tests we currently support and their state representations. The LastStateTest and StateTest are used to support discrete finite variable systems such as CTBN, as we will use in Sec. 5 and in experiments. Note the EventCountTest was the only supported test in the original PCIM paper. We can see that for tests that only depend

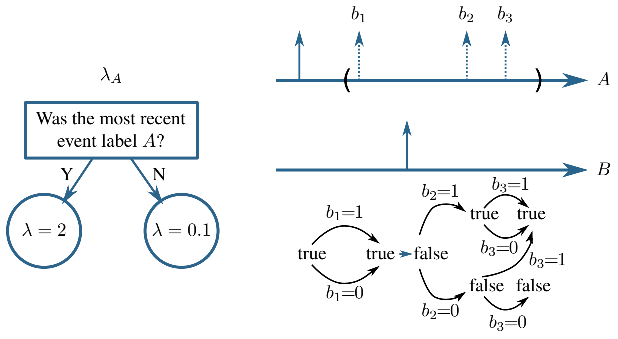


Figure 4: Dotted events are the virtual events that we sample as binary variables (b_i is 1 if event i is kept). The state diagram below the trajectory indicates the state of the test as we diverge (keep or drop a virtual event). Though there are 2^3 possible configurations, state merges can reduce the exponentially increasing complexity to linear in this case.

on the *current* time (i.e. TimeTest), the diverging history does not affect them, so no state is needed. For Markovian tests (LastEventTest and LastStateTest), we only need a Boolean variable. For the non-Markovian test (EventCountTest), the number of possible states does grow exponentially with the number of virtual events maintained in the queue. This is the best we can do and still be exact. It is much better than growing with the number of all virtual events. However, note that commonly $lag2 = 0$ and n is a small number. In this case, the state space size at any point is bounded as $\binom{m'}{n}$, where m' is the maximum number of sampled events in any time interval of duration $lag1$ (which is upper bounded by m). If n is 1, this is linear in the number of samples generated in during $lag1$ time units.

As noted above, if the test is not related to the sampled event (for example, in sampling event $l = A$ with test "are there $\geq$3 B events in the last 5 seconds"), the state of the test is null. This is because the evidence and sampled values for B (not the current variable for Gibbs sampling) can answer this test without reference to samples of l.

See Alg. 1 for the algorithm description for resampling event l. The complete algorithm iterates this procedure for each event label to get a new sample. The helper function UpdateState(s,b,t) returns the new state given the old state (s), the new time (t), and whether an event occurs at t (b). SampProbMap(M) takes a mapping from objects to positive values (M) and randomly returns one of the objects with probability proportional to the associate value. AddtoProbMap(M,o,p) checks to see if o is in M. If so, it adds p to the associated probability. Otherwise, it adds the mapping $o \rightarrow p$ to M.

4.7 Extended Example

Fig. 5 shows an example of resampling the events for label A on the unobserved interval $[0.8, 3.5)$. On the far left is

Table 1: Tests and their corresponding state representations.

Test	Example	State Representation	Property
TimeTest	Is the time between 6am and 9am?	Null	independent of b
LastEventTest	Is the last event A?	Boolean	Markovian
EventCountTest	Are there $>= n$ A event in $[t - lag1, t - lag2]$?	A queue maintaining all the times of A between $[t-lag2, t]$, and the most recent n events between $[t - lag1, t - lag2]$.	Non-Markovian
LastStateTest	Is the last sublabel of var $A=0$?	Boolean	Markovian
StateTest	Is the current sublabel of var $A=0$?	Null	independent of b

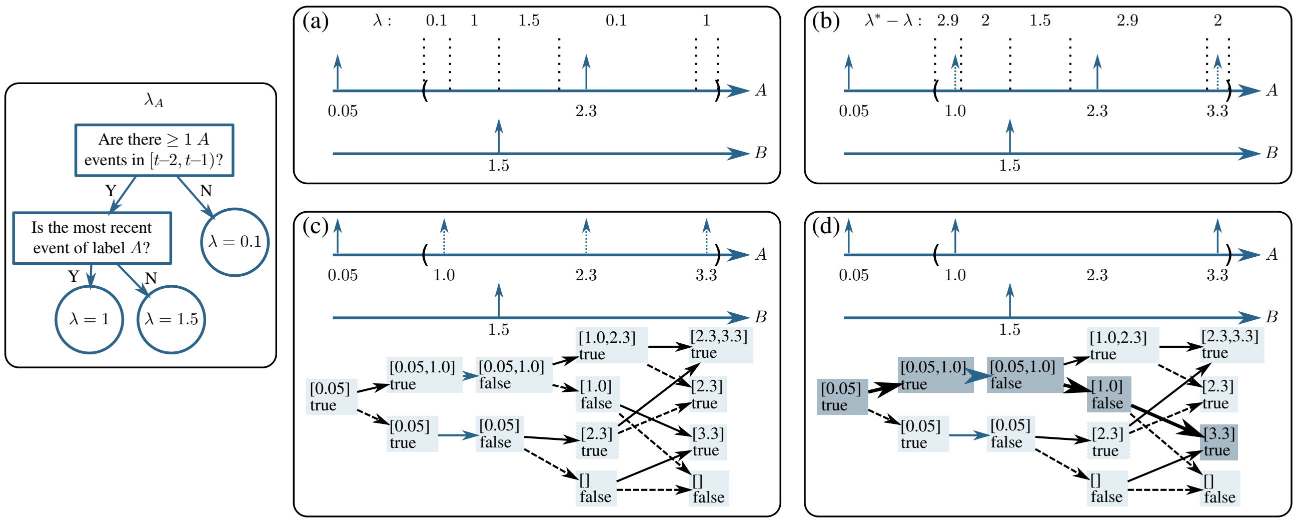


Figure 5: Extended Example, see Section 4.7

the PCIM rate tree for event A. Box (a) shows the sample from previous iteration (single event at 2.3). Dashed lines and λ show the piecewise-constant intensity function given the sample. Box (b) shows the sampling of virtual events. For this case $\lambda^* = 3$ for all time. $\lambda^* - \lambda$ is the rate for virtual events. The algorithm samples from this process, resulting in two virtual events (dashed). In box (c) all events become potential events. The state of the root test is a queue of recent events. The state of the other test is Boolean (whether A is more recent). On the bottom is the lattice of joint states over time. Solid arrows indicate $b_i = 1$ (the event is kept). Dash arrows indicate $b_i = 0$ (the event is dropped). Each arrow's weight is as per Eq. 4. The probability of a node is the sum over all paths to the node of the product of the weights on the path (calculated by dynamic programming). In box (d) a single path is sampled with backward sampling, shown in bold. This path corresponds to keeping the first and last virtual events and dropping the middle one.

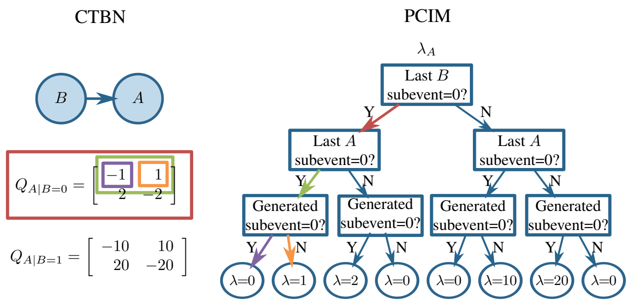


Figure 6: Explicit conversion from a CTBN to a PCIM by using specific tests. Only rates for variable A shown. The colored arrows and boxes show one-to-one correspondence of a path in the tree and an entry in the rate matrix of CTBN. Diagonal elements in the CTBN are redundant and do not need to be represented in the PCIM.

5 Representing CTBNs as PCIMs

A non-Markovian PCIM is more general than the Markovian CTBN model. We can, therefore represent a CTBN using a PCIM. In this way, we can extend PCIMs and we can compare our PCIM method with existing methods for CTBNs.

We associate a PCIM label with each CTBN variable. We also augment the notion of a PCIM label to include a sublabel. For each CTBN variable, its PCIM label has one sublabel for each state of the CTBN variable. Therefore, a

Algorithm 1 Resampling event l

```
input: The previous trajectory ($x_l$,$Y_l$)
output: The newly sampled $x_l^{'}$
1: for each unobserved interval for $l$ do
2:     Find piecewise constant $\lambda^*(t|h)$ using $Y_l$
3:     Find piecewise constant $\lambda(t|h)$ using $x_l$, $Y_l$
4:     Sample virtual events $v_l$ with rate $\lambda^*(t|h) - \lambda(t|h)$
5: Let $z_l = x_l \cup v_l$, $m = |z_l|$, and $s_0$ be the initial state
6: AddtoProbMap($S_0$,$s_0$,1.0)
7: for $i \leftarrow 1$ to $m$ do
8:     for each $\{(s_{i-1}, \cdot) \rightarrow p\}$ in $S_{i-1}$ do
9:         $p_{keep} = p(E_{i-1:i}, b_i = 1 \mid s_{i-1}, E_{1:i-1})$
10:        $p_{drop} = p(E_{i-1:i}, b_i = 0 \mid s_{i-1}, E_{1:i-1})$
11:        $s_i^{keep} \leftarrow$ UpdateState($s_{i-1}$, true, $z_{l,i}$)
12:        $s_i^{drop} \leftarrow$ UpdateState($s_{i-1}$, false, $z_{l,i}$)
13:        AddtoProbMap($S_i$,($s_i^{keep}$, $z_{l,i}$), $p \times p_{keep}$)
14:        AddtoProbMap($S_i$,($s_i^{drop}$, $\emptyset$), $p \times p_{drop}$)
15:        AddtoProbMap($T_i(s_i^{keep})$, $(s_{i-1}, z_{l,i})$, $p \times p_{keep}$)
16:        AddtoProbMap($T_i(s_i^{drop})$, $(s_{i-1}, \emptyset)$, $p \times p_{drop}$)
17: Update $S_m$ by propagating until ending time
18: $x_l^{'} \leftarrow \emptyset$ and $(s_m^{'}, t) \leftarrow$ SampProbMap($S_m$)
19: if $t \neq \emptyset$ then $x_l^{'} \leftarrow x_l^{'} \cup \{t\}$
20: for $i \leftarrow m-1$ to 1 do
21:     $(s_i^{'}, t) \leftarrow$ SampProbMap($T_{i+1}(s_{i+1}^{'})$)
22:     if $t \neq \emptyset$ then $x_l^{'} \leftarrow x_l^{'} \cup \{t\}$
23: return $x_l^{'}$
```

PCIM event with label X and sublabel x corresponds to a transition of the CTBN variable X from its previous value to the value x. The PCIM trees' tests can also check the sublabel associated with the possible event.

We augment the auxiliary Gibbs sampler to not only sample which virtual events are kept, but also which sublabel is associated with each. This involves modifying the b_i variables from the previous section to be multi-valued. Otherwise, the algorithm proceeds the same way.

The last two tests in Tbl. 1 are explicitly for this type of sublabelled event model. We can use them to turn a conditional intensity matrix from the CTBN into a PCIM tree. Fig. 6 shows the conversion of the "twonode" model.

6 Experiments

We implement our method as part of an open source code base, and all the code and data will be publicly available.

We perform two sets of experiments to validate our method. First we perform inference with our method on both Markovian and non-Markovian models, and compare the result with the ground-truth statistics. For both we show our result converges to the correct result. Ours is the first that can successfully perform inference tasks on non-Markovian PCIMs. For the second set of experiments, we use ThinnedGibbs in EM for both parameter estimation and structural learning for a non-Markovian PCIM. Our inference algorithm can indeed help producing models that achieve higher data likelihood on holdout test data than several baseline methods.

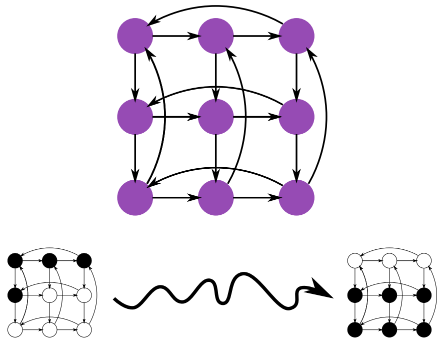


Figure 7: The toroid network and observed patterns [El-Hay et al., 2010].

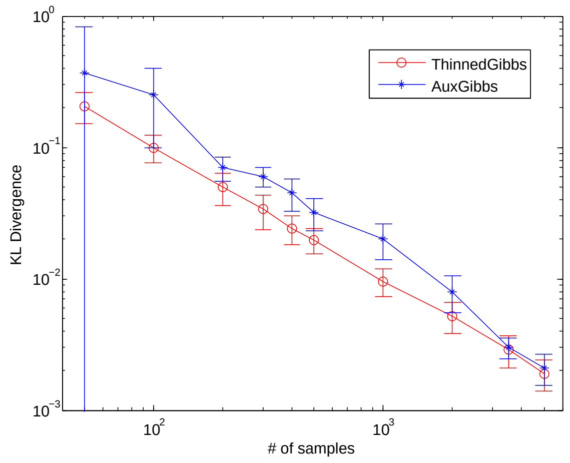


Figure 8: Number of samples versus KL divergence for the toroid network. Both axes are on a log scale.

6.1 Verification on the Ising Model

We first evaluate our method, ThinnedGibbs, on a network with Ising model dynamics. The Ising model is a well-known interaction model with applications in many fields including statistical mechanics, genetics, and neuroscience. This is a Markovian model and has been tested by several existing inference methods designed for CTBNs.

Using this model, we generate a directed toroid network structure with cycles following [El-Hay et al., 2010]. Nodes can take values -1 and 1, and follow their parents' states according to a coupling strength parameter (β). A rate parameter (τ) determines how fast nodes toggle between states. We test with $\beta = 0.5$ and $\tau = 2$. The net-

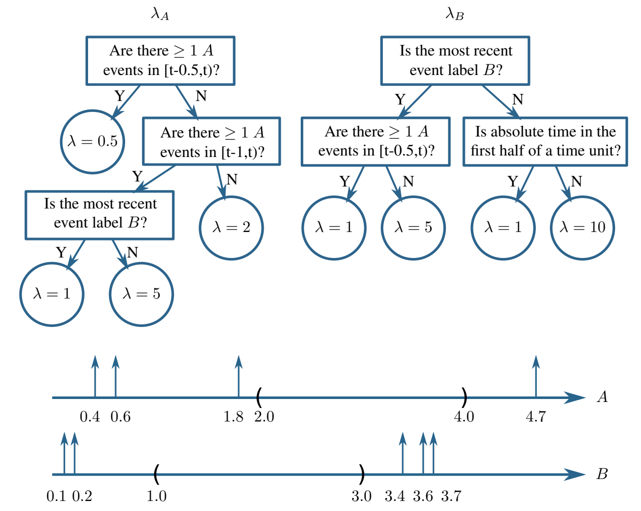

λ_A
Are there ≥ 1 A events in [t-0.5,t)?
Y
N
$\lambda = 0.5$
Are there ≥ 1 A events in [t-1,t)?
Y
N
Is the most recent event label B?
Y
N
$\lambda = 1$
$\lambda = 5$
$\lambda = 2$
λ_B
Is the most recent event label B?
Y
N
Are there ≥ 1 A events in [t-0.5,t)?
Y
N
$\lambda = 1$
$\lambda = 5$
Is absolute time in the first half of a time unit?
Y
N
$\lambda = 1$
$\lambda = 10$
A
0.4 0.6 1.8 2.0 4.0 4.7
B
0.1 0.2 1.0 3.0 3.4 3.6 3.7

Figure 9: Non-Markovian PCIM and evidence. The ending time is 5.

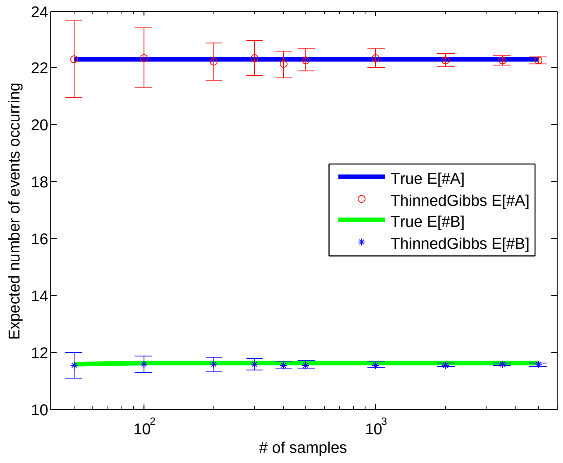


Figure 10: Number of samples versus the inferred expected number of events. The horizontal axis is on a log scale.

work and the evidence patterns are shown in Fig. 7. The nodes are not observed between $t = 0$ and $t = 1$. We query the marginal distribution of nodes at $t = 0.5$ and measure the sum of the KL-divergences of all marginals against the ground truth. We compare with the state-of-the-art CTBN Auxiliary Gibbs method [Rao and Teh, 2013]. Other existing methods either produce similar or worse results [Celikkaya and Shelton, 2014]. We vary the sample size between 50 and 5000, and set the burn-in period to be 10% of this value. We run the experiments for 100 times, and plot the means and standard deviations.

Results in Fig. 8 verify that our inference method indeed produces results that converge to the true distribution. Our method reduces to that of [Rao and Teh, 2013] in this Markovian model. Differences between the two lines are due to slightly different initializations of the Gibbs Markov chain and not significant.

6.2 Verification on a Non-Markovian Model

We further verify our method on a much more challenging non-Markovian PCIM (Fig. 9). This model contains several non-Markovian EventCountTests. We have observations for event A at $t = 0.4, 0.6, 1.8, 4.7$ and for event B at $t = 0.1, 0.2, 3.4, 3.6, 3.7$. Event A is not observed on $[2.0, 4.0)$ and event B is not observed on $[1.0, 3.0)$.

In produce ground truth, we discretized time and converted the system to a Markovian system. Note that because the time since the last A event is part of the state, as the discretization becomes finer, the state space increases. For this small example, this approach is just barely feasible. We continued to refine the discretization until the answer stabilized. The ground-truth expected total number of A events between $[0, 5]$ is 22.3206 and the expected total number of B events is 11.6161. That is, there are about 18.32 A events and 6.62 B events in the unobserved areas. Note that if the evidence is changed to have no events these numbers drop to 1.6089 and 8.6866 respectively and if the evidence after the unobserved intervals is ignored the expectations are 22.7183 and 8.6344 respectively. Therefore the evidence (both before and after the unobserved intervals) is important to incorporate in inference.

We compare our inference method to the exact values, again varying the sample size between 50 and 5000 and setting the burn-in period to be 10% of this value. We ran the experiments 100 times and report the mean and standard deviation of the two expectations. Our sampler has very small bias and therefore the average values match the true value almost exactly. The variance decreases as expected, demonstrating the consistent nature of our method. See Fig. 10. We are not aware of existing methods that can perform inference on this type of model to which we could compare.

6.3 Parameter Estimation and Structural Learning

We further test ThinnedGibbs by using it in EM, for both parameter estimation (given the tree structure, estimate the rates in the leaves), and structural learning (learn both the structure and rates). We use Monte Carlo EM that iterates between two steps: First, given a model we generate samples conditioned on evidence with ThinnedGibbs. Second, given the samples, we treat them as complete trajectories and perform parameter estimation and structural learning, which is efficient for PCIM. We initialize the model from the partial trajectories, assuming no events occur in the unobserved intervals. EM terminates when the parameters of PCIMs in two consecutive iterations are stable (all rates change less than 10% from the previous ones), or the num-

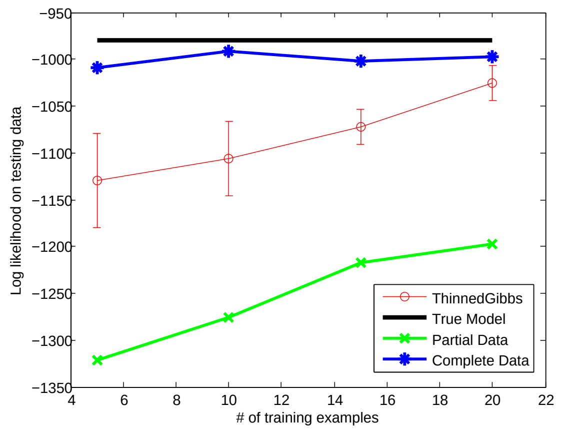


Figure 11: Parameter estimation. Testing log-likelihood as a function of the number of training samples.

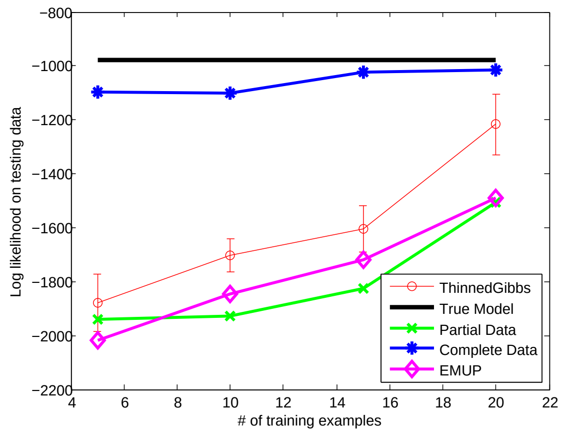


Figure 12: Structure and parameter estimation. Testing log-likelihood as a function of the number of training examples.

ber of iterations surpasses 10. For structural learning, the structure needs to be the same between iterations.

We use the model in Fig. 1 and generate complete trajectories for time range $[0, 10)$. We vary the number of training samples (5, 10, 15, and 20) and use a fixed set of 100 trajectories as the testing data. For each training size, we use the same the training data for all algorithms and runs. We randomly generate an unobserved interval with length $0.6 \times T$ for both event labels. For each training sample, ThinnedGibbs fills it in to generate a new sample after burning in 10 steps. For each configuration, we run ThinnedGibbs for 5 times. We measure the data likelihood of the holdout testing data on the learned models.

For parameter estimation, we compare with the true model that generated the data, the model learned with only partial data in which we assume no events happened during unseen intervals (Partial Data), and a model learned with complete training data (Complete Data). The results are summarized in Fig. 11. We can see that the model learned by EM algorithm using ThinnedGibbs can indeed produce significantly higher testing likelihood than using only partial data. Of course, we do not do as well as if none of the data had been hidden (Complete Data).

If learning the structure, there is one other possibility: We could use the original fast PCIM learning method, but indicate (by new event labels) when an unobserved interval starts and stops. We augment the bank of possible decisions to include testing if each pseudo-events have occurred most recently. In this way, the PCIM directly models the process that obscures the data. Of course, at test time, branches modeling such unobserved times are not used. Such model should serve as a better baseline than learning from partially observed data, because it can potentially learn unobserved patterns and only use the dependencies in the observed intervals for a better model. We call this model EMUP (explicit modeling of unobserved patterns).

For structure learning, we fix the bank of possible PCIM tests as EventCountTests with $(l, n, lag1, lag2) \in \{A, B\} \times \{1, 2\} \times \{2, 3, 4, 5, 6\} \times \{0, 1, 2\}$ (omitting tests for which $lag1 \leq lag2$). For EMUP we also allow testing if currently in unobserved interval. The results are summarized in Fig. 12. We can see that EMUP does outperform models using only partial data. However, Structural EM with ThinnedGibbs still performs better. The performance gain is less than that in the parameter estimation task, probably because there are more local optimums for structural EM, especially with fewer training examples.

7 Discussion and Future Work

We proposed the first effective inference algorithm, ThinnedGibbs, for PCIM. Our auxiliary Gibbs sampling method effectively transforms a continuous-time problem into a discrete one. Our state-vector representation of diverging trajectories takes advantage of state merges and reduces complexity from exponential to linear for most cases. We build the connection between PCIM and CTBN, and show our method generalizes the state-of-art inference method for CTBN models. In experiments we validate our idea on non-Markovian PCIMs, which is the first to do so.

Our method converges to the exact conditional distribution. If the true state of the model indeed grows exponentially, the complexity of ThinnedGibbs follows. We believe this technique could also be applied to other non-Markovian processes. The challenge lies in computing the forward-pass likelihoods when the rate function is not piecewise-constant.

Acknowledgement

This work was supported by DARPA (FA8750-14-2-0010).

References

[Celikkaya and Shelton, 2014] Celikkaya, E. B. and Shelton, C. R. (2014). Deterministic anytime inference for stochastic continuous-time Markov processes. In *ICML*. 8

[Cohn et al., 2009] Cohn, I., El-Hay, T., Kupferman, R., and Friedman, N. (2009). Mean field variational approximation for continuous-time bayesian networks. In *UAI*. 2

[Dean and Kanazawa, 1988] Dean, T. and Kanazawa, K. (1988). Probabilistic temporal reasoning. In *AAAI*. 1

[Du et al., 2013] Du, N., Song, L., Gomez-Rodriguez, M., and Zha, H. (2013). Scalable influence estimation in continuous-time diffusion networks. In *NIPS*. 2

[El-Hay et al., 2010] El-Hay, T., Cohn, I., Friedman, N., and Kupferman, R. (2010). Continuous-time belief propagation. In *ICML*. 2, 7

[Fan et al., 2010] Fan, Y., Xu, J., and Shelton, C. R. (2010). Importance sampling for continuous time Bayesian networks. *Journal of Machine Learning Research*, 11(Aug):2115–2140. 2

[Golightly and Wilkinson, 2011] Golightly, A. and Wilkinson, D. J. (2011). Bayesian parameter inference for stochastic biochemical network models using particle Markov chain Monte Carlo. *Interface Focus*. 2

[Grassmann, 1977] Grassmann, W. K. (1977). Transient solutions in Markovian queueing systems. *Computers & Operations Research*, 4(1):47–53. 2, 3

[Gunawardana et al., 2011] Gunawardana, A., Meek, C., and Xu, P. (2011). A model for temporal dependencies in event streams. In *NIPS*. 1, 2, 3

[Lewis and Shedler, 1979] Lewis, P. A. W. and Shedler, G. S. (1979). Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413. 1, 2, 3

[Linderman and Adams, 2014] Linderman, S. W. and Adams, R. P. (2014). Discovering latent network structure in point process data. In *ICML*. 2

[Nodelman et al., 2002] Nodelman, U., Shelton, C. R., and Koller, D. (2002). Continuous time bayesian networks. In *UAI*. 2

[Parikh et al., 2012] Parikh, A., Gunawardana, A., and Meek, C. (2012). Cojoint modeling of temporal dependencies in event streams. In *UAI Workshops*. 2

[Rajaram et al., 2005] Rajaram, S., Graeoel, T., and Herbrich, R. (2005). Poisson-networks: A model for structured point process. In *AIStats*. 2

[Rao and Teh, 2011] Rao, V. and Teh, Y. W. (2011). Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In *UAI*. 2, 3

[Rao and Teh, 2013] Rao, V. and Teh, Y. W. (2013). Fast MCMC sampling for Markov jump processes and extensions. *Journal of Machine Learning Research*, 14:3207–3232. arXiv:1208.4818. 2, 3, 4, 8

[Saito et al., 2009] Saito, K., Kimura, M., Ohara, K., and Motoda, H. (2009). Learning continuous-time information diffusion model for social behavioral data analysis. In *ACML*, pages 322–337. 2

[Simma and Jordan, 2010] Simma, A. and Jordan, M. (2010). Modeling events with cascades of poisson processes. In *UAI*. 2

[Weiss and Page, 2013] Weiss, J. and Page, D. (2013). Forest-based point processes for event prediction from electronic health records. In *ECML-PKDD*. 2

Appendix E

Social Grouping for Multi-target Tracking and Head Pose Estimation in Video

Appeared as

Zhen Qin and Christian R. Shelton. Social Grouping for Multi-target Tracking and Head Pose Estimation in Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10), 2082–2095, 2016.

Social Grouping for Multi-target Tracking and Head Pose Estimation in Video

Zhen Qin and Christian R. Shelton

Abstract—Many computer vision tasks are more difficult when tackled without contextual information. For example, in multi-camera tracking, pedestrians may look very different in different cameras with varying pose and lighting conditions. Similarly, head direction estimation in high-angle surveillance video in which human head images are low resolution is challenging. Even humans can have trouble without contextual information. In this work, we couple novel contextual information, social grouping, with two important computer vision tasks: multi-target tracking and head pose/direction estimation in surveillance video. These three components are modeled in a probabilistic formulation and we provide effective solvers. We show that social grouping effectively helps to mitigate visual ambiguities in multi-camera tracking and head pose estimation. We further notice that in single-camera multi-target tracking, social grouping provides a natural high-order association cue that avoids existing complex algorithms for high-order track association. In experiments, we demonstrate improvements with our model over models without social grouping context and several state-of-art approaches on a number of publicly available datasets on tracking, head pose estimation, and group discovery.

Index Terms—Multi-target tracking, multi-camera tracking, head pose estimation, social grouping, video analysis, context.

1 Introduction

It is difficult to achieve satisfactory results purely by using visual information for many computer vision tasks due to the inherent visual ambiguities in real-world images and videos. Take multi-camera tracking as an example. Pedestrians may look quite different under cameras with varying conditions. Another example is head pose estimation in high-angle surveillance video. (We focus on yaw angle estimation in such scenarios.) Human head images are usually of low resolution, which makes visual evidence unreliable (see Fig. 1). Thus, contextual information is needed for these tasks.

We introduce social grouping as one such context. Sociology research [29] shows that in natural scenes up to 70% of people walk in groups, possessing similar trajectories, speed, and destinations. These factors should help to disambiguate confusing tracking decisions in both single-camera (similar trajectories and speed) and multi-camera tracking (similar destinations). For example, in multi-camera tracking, the tracker usually finds it difficult to decide linking or splitting two detections, since one person usually looks quite different in two cameras. However, if the two detections are accompanied by another person, linking is preferred. It is also intuitively clear that when people form groups, their head directions are correlated, as they tend to look at each other or the same area of interest.

In this work, we provide a probabilistic framework with effective solvers to utilize social grouping for visual tracking and head pose estimation. The joint optimization of tracking and social grouping is modeled as a constrained nonlinear optimization problem, which results in steps involving standard fast procedures. Head pose estimation in groups is modeled as a graph labeling problem using a conditional random field (CRF) that allows exact convex learning and inference, with tractability supported by sociology research. The generality of our social grouping model makes it applicable to most existing tracklet linking and head pose estimation frameworks.

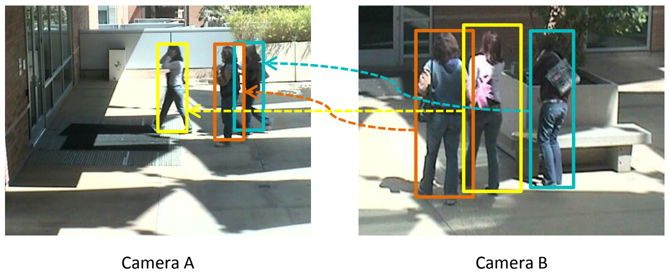
Our experiments show that social context can help in multi-target tracking and head pose estimation on real-world datasets. Of particular interest, social grouping provides a natural high-order cue for the single-camera multi-target tracking problem, while existing approaches usually depend on complex solvers to go beyond single-order association. Furthermore, social grouping is also an output of the complete system. Our model produces results that are comparative to or better than state-of-art methods on benchmark datasets (see Tbl. 1) on all three tasks (tracking, head pose estimation, and group discovery), though our model employs only simple motion and visual features.

Preliminary pieces of this work described the coupling of social grouping with single-camera [36] and multi-camera tracking [37]. In this paper, we also include head pose estimation and provide a unified view. In addition, we provide more comprehensive experimental results, including group discovery performance.

2 Related Work

Head pose estimation, group discovery, and especially multi-target tracking, have been extensively researched in the computer vision community. We focus on the literature that is most related to our work.

• Z. Qin and C. R. Shelton are with the Department of Computer Science and Engineering, University of California, Riverside, Riverside, CA, 92521.
E-mail: {zqin001, cshelton}@cs.ucr.edu
This work was supported by DARPA



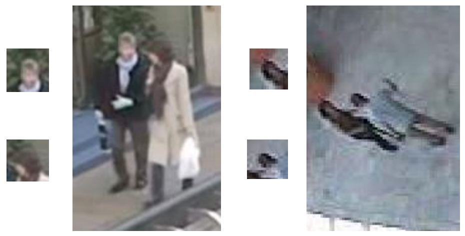


Fig. 1: (Left) Social grouping behavior not only generally exists in one scene, but also usually persists (with the same group members) across wide areas. (Right) Given head images alone, it is sometimes difficult for human beings to correctly identify head pose directions in challenging scenarios. Social context provides strong evidence for this difficult problem.

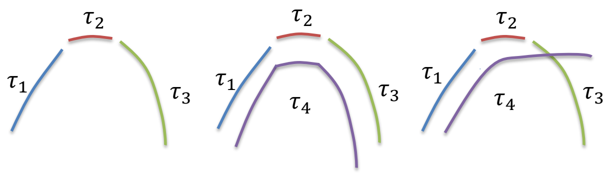


Fig. 2: (Left) Motion dependency problem for order-one association methods [48]: though $\tau_1 - \tau_2$ and $\tau_2 - \tau_3$ can be reasonably pairwise linked, the full trajectory is not probable. (Middle, Right) Social context from τ_4 gives strong evidence to disambiguate the dependency among tracks, indicating $\tau_1 - \tau_2 - \tau_3$ is probable (middle) or not (right).

Single-camera multi-target tracking. Multi-target tracking is a key step in many computer vision tasks, including visual surveillance, activity recognition, and scene understanding. Time-critical approaches usually use particle filtering algorithms for state estimation [51]. However, it is very difficult for such systems to handle long-term occlusions and detection failures. Thus recently, data association-based tracking (DAT, also known as the tracklet-linking problem) has dominated the research community. With the help of state-of-art tracklet extraction methods such as human detector approaches [25], researchers look at extended time periods and link conservatively extracted tracklets (short tracks) to recover full tracks. Many focus on how to obtain more reliable linking probabilities between tracklets [25][23][21]. To effectively infer the best matching given the affinity measurements among tracklets, different optimization methods such as the Hungarian algorithm [25][41], K-shortest path [4], MWIS [5], set-cover [46], min-cost flow [6], approximate dynamic programming [34], and continuous energy minimization [28] have been proposed. Some of them are shown to be equivalent to each other [19]. Importantly, these methods are mostly order-one methods, meaning that they optimize only pairwise similarities. This might lead to global inconsistencies. One typical problem is the motion dependency problem described in Fig. 2.

Yang et al. [48] employ a CRF model to mitigate the motion dependency problem for single tracks. Butt and Collins [6] use a relaxation to the min-cost network flow framework to explore higher-order smoothness constraints such as constant velocity. These models involve complex solvers and still possess limitations as they only address the motion dependency problem for single tracks: As shown in Fig. 2, the likelihood of one track with sudden motion change might depend on whether it is accompanied by a group member with a similar trajectory. Our model, on the other hand, models such scenarios by design, can be built upon simple solvers, and naturally helps higher-order tracking when coupling with social grouping information (modeled as a global spatial-temporal clustering procedure).

Multi-camera multi-target tracking. Multi-camera systems are ubiquitous, and a reliable multi-camera tracking system allows wide-area scene understanding. Researchers typically employ spatial-temporal and appearance cues to handover targets across cameras. For spatial-temporal information, Javed et al. [20] use a Parzen window density estimator to jointly model the inter-camera travel time intervals, locations of exit/entrances, and velocities of objects. Makris et al. [26] propose an unsupervised learning method to validate the camera network model. In terms of appearance similarity, Javed et al. [20] show that the Brightness Transfer Function (BTF) between cameras lies in a low dimensional subspace and proposes a method to learn them with labeled correspondences. A cumulative brightness transfer function (CBTF) is proposed by Prosser et al. [35] for mapping color between cameras using sparse training set. Kuo et al. [22] use Multiple Instance Learning (MIL) to learn a discriminative appearance affinity model online. The work by Orazio et al. [15] evaluates several BTFs and shows that they demonstrate similar behaviors and limitations. Our work, on the other hand, is the first to explore social grouping for the multi-camera tracking problem, which is more robust to changes in camera characteristics, viewpoints, and illumination conditions.

Head pose estimation. Head pose and gaze estimation is a long-studied area in computer vision and human computer interaction (HCI). It enables various applications such as human attention tracking and area or object of focus detection [27][1]. Most work focuses on head image classification where images possess reasonable resolutions and face landmarks are visible. Murphy-Chutorian and Trivedi [30] give an excellent review on diverse approaches towards this problem. Recent advances in this area include using part-based model [53]. In this work, we focus on head pose estimation in the common high-angle surveillance video, also known as head direction estimation [10] and coarse gaze estimation [3]. Compared to traditional pose estimation work, the visual features of head images are usually very weak considering their small sizes, thus methods requiring face landmarks are not applicable. This problem is usually modeled as a regression problem (though discretized classes sometimes serve as an intermediate step, due to the ease of dealing with discrete labels over real-valued angles [10] [38]. Our work follows this approach), where angle difference between prediction and annotation, instead of classification accuracy, is measured, because of the difficulty of accurate class labeling [13] and the contiguity of nearby classes/angles in the feature space. Most work still focuses on feature extraction and estimation based on head images alone: Robertson and Reid [38] explore skin color feature. Tosato et al. [44] explore covariance features. The histogram of gradient (HoG) [12] is popular recently [3][10]. Support Vector Machine (SVM), SVM Regressor, Neural Networks, Decision Trees, and Nearest Neighbor classifiers are among classifiers/regressors applied [44][38][31]. The recent representative work by Benfold and Reid [3] employs structured learning, proposing a CRF for head pose estimation. Chamveha et al. [9] employ spectral clustering for scene adaptation. Chen and Odobez [10] couple head direction estimation with body pose in a general kernel learning framework. However, all these existing work only consider individuals. We consider general social tendency and repellence beyond individuals.

Group discovery. Social discovery has also drawn much attention in the computer vision community recently [47][18][8][40]. Ge et al. [18] infer social groups given a tracking result. By contrast, we perform grouping and tracking jointly. Chamveha et al. [8] use attention cue to help discovering groups, while we perform grouping first to aid head pose estimation. This is because we note that in challenging scenarios, head pose estimation can be more difficult than group discovery (Ge et al. [18] also note that trajectory information alone is enough to yield substantial agreement with human for the grouping task).

Socially-aware computer vision. Social context has been explored in a number of computer vision problems. For tracking, Pirsiavash et al. [32] proposed a more effective dynamic model based on social information, Pirsiavash et al. [33] and Yamaguchi et al. [47] infer grouping for better trajectory prediction and behavior prediction respectively. Bazzani et al. [2] focuses on tracking groups and Chen et al. [11] considers local group consistency. Ours is the first to consider social grouping context for the data association-based multi-target tracking and head pose estimation problem.

3 Social Grouping for Multi-target Tracking and Head Pose Estimation

We first introduce our notation and the probabilistic formulation of utilizing social grouping for multi-target tracking and head pose estimation as two maximum a posteriori (MAP) problems.

3.1 Notation

The input of our system is a set of n tracklets (possibly including false alarms) $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ within a time interval [0, T], extracted by methods described in Sec. 6.2. Each tracklet τ_i is a sequence of short descriptions of a single target across the time interval [t_i^{start}, t_i^{finish}]. Such descriptions include the position and size of target (for the tracking problem), and the position and size of the pedestrian head (for the head pose estimation problem). In particular we let $a_i(t)$ be the camera (discrete camera labels) and $l_i(t)$ be the position (discrete pixel coordinates in the image) of τ_i at time t. We abuse $l_i(t)$ to denote both pedestrian and head positions.

The task of multi-target tracking is to determine which tracklets correspond to the same target, which can be represented as a binary correspondence matrix ϕ:

$$\phi_{i,j} = \begin{cases} 1 & \text{if tracklet } j \text{ immediately follows tracklet } i, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

with the added constraints that $\sum_j \phi_{i,j} = 1$ and $\sum_i \phi_{i,j} = 1$, indicating each tracklet should only follow and be followed by one other tracklet (except for the first and last tracklets of each track, addressed by virtual starting and ending tracklets in Sec. 4.4.1). We let Φ be the set of valid correspondence matrices.

For social grouping evaluation, we model it as a clustering problem and assume people form K groups, where K is unknown. Within each group, there is a group mean trajectory (a sequence of image coordinates) G_k, with $G = \{G_1, G_2, \dots, G_K\}$. ψ denotes a binary social grouping assignment matrix:

$$\psi_{i,k} = \begin{cases} 1 & \text{if tracklet } i \text{ is assigned to group } k, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Again there is an added constraint that $\sum_k \psi_{i,k} = 1$ and we let Ψ be the set of valid social grouping matrixes.

For group head pose estimation, we will process each group independently at every time point so we drop the time stamp here. Let C denote the number of individuals in a group, Y denote the head directions of everyone in the group, Υ denote the head directions of all head images in the scene, X denote any existing unary evidence for individuals (such as image values or walking direction; there are M such features), and L denote the pedestrians' head locations. Let y_j and l_j be the head direction and location of the jth person, and x_j^i be the ith unary evidence for the jth person. Thus $Y = \{y_1, \dots, y_C\}$, $L = \{l_1, \dots, l_C\}$, $X^i = \{x_1^i, \dots, x_C^i\}$, and $X = \{X^1, \dots, X^M\}$. Information of X and L can be extracted from tracklet descriptions.

3.2 The Probabilistic Model Formulation

The inference of tracking, group discovery, and head pose estimation given inputs can be modeled as two maximum a posteriori (MAP) problems:

$$(\phi^*, \psi^*, G^*) = \underset{\phi \in \Phi, \psi \in \Psi, G}{\arg\max}\ P(\phi, \psi, G | \tau) \tag{3}$$

and

$$\Upsilon^* = \arg\max_{\Upsilon} P(\Upsilon|\phi, \psi, G, \tau). \tag{4}$$

In our work, the input to the second problem is the output of the first problem. Thus a single forward filtering of these two steps would output all desired information (tracking, group discovery, head pose estimation).

4 Coupling Social Grouping with Multi-target Tracking

We model the first MAP problem, $P(\phi, \psi, G|\tau)$, as

$$\begin{aligned} P(\phi, \psi, G|\tau) &\propto P(\phi, \psi, G, \tau) \\ &= P(G)\ P(\tau, \psi|G)\ P(\phi|\tau, \psi, G) \\ &= P(G)\ P(\tau, \psi|G)\ P(\phi|\tau, \psi), \end{aligned} \tag{5}$$

assuming group trajectories do not affect tracklet linking given grouping assignments. Next we explain each component of this model and the optimization algorithm.

4.1 Social Grouping as K-means Clustering

$P(\tau, \psi|G)$ is the data likelihood function of the probabilistic interpretation of clustering algorithms such as K-means clustering. We have

$$P(\tau, \psi|G) \propto \prod_{i,k|\psi_{ik}=1} P(\tau_i|G_k), \tag{6}$$

assuming trajectories for each individual are independent from each other given group mean trajectories (a similar assumption is made in general K-means clustering). $P(\tau_i|G_k)$ is the likelihood that tracklet i comes from group k, which we decompose across time as

$$P(\tau_i|G_k) = \prod_{t=t_i^{start}}^{t_i^{finish}} P(a_i(t)|G_k)\ P(l_i(t)|a_i(t), G_k). \tag{7}$$

$P(a_i(t)|G_k)$ is the probability that group k appears at camera $a_i(t)$, a parameter of the model for group k which we denote as $b_{k,a}(t)$. $P(l_i(t)|a_i(t), G_k)$ is the probability that at time t, a member of the group in camera $a_i(t)$ will appear at position $l_i(t)$, which we model as a Gaussian centered around the mean $u_{k,a}(t)$, the position for group k in camera a at time t, also a parameter of the model for group k. We use a fixed variance for all such Gaussians.

Notice that here we provide a general formulation for the multi-camera scenario. When it is the single-camera case, Eq. 7 can be significantly simplified ($P(a_i(t)|G_k)$ can be dropped).

4.2 Socially Constrained Multi-target Tracking

$P(\phi|\tau, \psi)$ measures the probability of tracklet linking (or track handover in the multi-camera case) given the social group information. Compared to traditional tracking methods, this adds a group constraint that if two tracklets are linked (they are the same person), they belong to the same group (one group per person):

$$\begin{aligned} P(\phi|\tau, \psi) = &\prod_{i|\forall m, \phi_{m,i}=0} P_{init}(\tau_i) \prod_{j|\forall m, \phi_{j,m}=0} P_{term}(\tau_j) \\ &\times \prod_{i,j|\phi_{i,j}=1} \begin{cases} P_{link}(i,j) & \text{if } \forall k, \psi_{i,k} = \psi_{j,k}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{8}$$

where $P_{init}(\tau_i)$ is the likelihood of τ_i being an initial tracklet, and $P_{term}(\tau_j)$ the likelihood of τ_j being the last tracklet. $P_{link}(i,j)$ is the likelihood that tracklet j is the first instance following tracklet i. These probabilities are the affinity model; any standard cues from the literature can be used (see Sec. 6.3).

4.3 A Simple Social Group Model

We model the probability of social groups as

$$P(G) \propto e^{-\kappa|G|}, \tag{9}$$

penalizing large numbers of social groups to avoid overfitting (such as placing each person in a separate group). Note that other heuristics are also applicable. Our choice is intuitive and results in a simple linear penalty in the optimization space, with its effectiveness validated in experiments.

4.4 Joint Optimization of Social Grouping and Multi-target Tracking

This section introduces the joint optimization of tracking and social grouping ($P(G)$, $P(\tau, \psi|G)$, and $P(\phi|\tau, \psi)$ in Eq. 5) as a constrained nonlinear optimization framework, which we call SGB (Social Grouping Behavior) algorithm.

We first reformulate the joint optimization of social grouping and multi-target tracking in the negative log space and achieve clean formulations. Then we introduce an effective optimization framework that can result in simple existing methods.

4.4.1 Optimization Reformulation

We perform the joint optimization of tracking and social grouping in the negative log-likelihood space (a minimization problem). Ignoring an additive constant from the proportionality in Eq. 9,

$$-\ln P(G) = \kappa|G|. \tag{10}$$

This term is in charge of selecting the number of groups and serves as the outer loop of optimization. Ignoring a similar additive constant, for $P(\tau, \psi|G)$ (Eq. 6), we have $-\ln P(\tau, \psi|G) = \sum_{i,k|\psi_{ik}=1} D(\tau_i, G_k) =$

$$\sum_{i,k|\psi_{ik}=1} \sum_{t=t_i^{start}}^{t_i^{finish}} -\alpha \ln b_{k,a_i(t)}(t) + \beta \big|l_i(t) - u_{k,a_i(t)}(t)\big)\big|^2 \tag{11}$$

from Eq. 7 where α and β are weights relating to the variance of the Gaussian. For simplicity, we define $D(\tau_i, G_k)$

to be the "distance" of tracklet i from group k as above. In the single-camera case, the distribution $b_{k,a}(t)$ is degenerate and drops out of the equation.

$P(\phi|\tau,\psi)$ (Eq. 8) can be transformed to an assignment problem by defining a $2n \times 2n$ tracklet linking matrix

$$H = \left(\begin{array}{c|c} H^{link}_{n\times n} & H^{term}_{n\times n} \\ \hline H^{init}_{n\times n} & \infty_{n\times n} \end{array} \right) \tag{12}$$

with $H^{link}_{i,j} = -\ln P_{link}(i,j)$, $H^{init}_{i,i} = -\ln P_{init}(\tau_i)$, $H^{term}_{i,i} = -\ln P_{term}(\tau_i)$ and infinity $(-\ln 0)$ elsewhere (including all diagonal elements). The virtual tracklets are introduced to handle track initializations and terminations. Eq. 8 is 0 if any assignments violate the constraint that linked tracklets must be in the same social group. Therefore, if we add this as a constraint: $\forall i,j,k\ \phi_{i,j}(\psi_{i,k}-\psi_{j,k}) = 0$, the resulting equation can be written in terms of H:

$$-\ln P(\phi|\tau,\psi) = \sum_{i,j} \phi_{i,j} H_{i,j} \tag{13}$$

Our optimization's outer loop tries different numbers of social groups ($P(G)$). Inside (optimizing $P(\tau,\psi|G)$ and $P(\phi|\tau,\psi)$), we can drop Eq. 10 and minimize the sum of Eq. 13 and Eq. 11 with the above constraint:

$$\begin{aligned} &\min_{\phi\in\Phi,\psi\in\Psi,G} && \sum_{i,j}\phi_{i,j}H_{i,j} + \sum_{i,k}\psi_{i,k}D(\tau_i,G_k) \\ &\text{s.t.} \quad \forall i,j,k && \phi_{i,j}(\psi_{i,k}-\psi_{j,k}) = 0. \end{aligned} \tag{14}$$

We call Eq. 14 the primal problem.

4.4.2 A Two-stage Alternating Minimization Algorithm

We use a two-stage iterative alternative optimization algorithm to solve the constrained nonlinear optimization problem in Eq. 14. The Lagrangian is

$$\begin{aligned} L(\phi,\psi,G,\mu) =& \sum_{i,j}\phi_{i,j}H_{i,j} + \sum_{i,k}\psi_{i,k}D(\tau_i,G_k) \\ &+ \sum_{i,j,k}\mu_{i,j,k}\phi_{i,j}(\psi_{i,k}-\psi_{j,k}), \end{aligned} \tag{15}$$

where the μs are the Lagrange multipliers. The dual of this problem is

$$\begin{aligned} &\max_{\mu}\ q(\mu) \\ \text{where} \quad & q(\mu) = \min_{\phi\in\Phi,\psi\in\Psi,G} L(\phi,\psi,G,\mu). \end{aligned} \tag{16}$$

The resulting correspondence ϕ of the optimization is the output of the method. For a fixed μ, let

$$(\phi^\mu,\psi^\mu,G^\mu) = \operatorname*{arg\,min}_{\phi\in\Phi,\psi\in\Psi,G} L(\phi,\psi,G,\mu). \tag{17}$$

To solve Eq. 16, we use a quasi-Newton strategy with limited-memory BFGS updates and Wolfe line search conditions guided by the subgradient [39]:

$$\left.\frac{\partial q}{\partial \mu_{i,j,k}}\right|_{\mu} = \phi^\mu_{i,j}(\psi^\mu_{i,k}-\psi^\mu_{j,k}). \tag{18}$$

To calculate the subgradient, we use a two-stage block coordinate-minimization algorithm to solve Eq. 17. The first stage minimizes over ϕ (the tracklet correspondence result) from Eq. 15 with ψ and G fixed:

$$\phi^\mu = \operatorname*{arg\,min}_{\phi\in\Phi}\sum_{i,j}\phi_{i,j}[H_{i,j} + \sum_k \mu_{i,j,k}(\psi_{i,k}-\psi_{j,k})]. \tag{19}$$

This amounts to adding a penalty term to the matrix scores (compare with Eq. 13). So Eq. 19 is a standard assignment problem and can be efficiently solved by the Hungarian algorithm (or any algorithm designed for tracklet linking).

The second stage minimizes Eq. 15 over ψ and G, with ϕ fixed: $(\psi^\mu, G^\mu) =$

$$\operatorname*{arg\,min}_{\psi\in\Psi,G}\sum_{i,k}\psi_{i,k}[D(\tau_i,G_k) + \sum_j(\mu_{i,j,k}\phi_{i,j} - \mu_{j,i,k}\phi_{j,i})]. \tag{20}$$

This amounts to a standard K-means clustering problem. If the "centers," G, are fixed, the assignments, ψ, are made to minimize the augmented distance. When the assignments are fixed, the centers can be placed to minimize their distances to the captured points. Several initial group assignments are tried, as K-means converges to local minimum. The output of the one with the minimum value for Eq. 16 for one specific $|G|$ is maintained. At the end, we add the linear penalty of $|G|$ indicated by Eq. 10 and the outer loop (over $|G|$) selects the solution with the minimal negative log-likelihood score. See Alg. 1 for details.

Our method can be viewed as approximate max-product on the graph $G-\psi-\phi$ (in which the constraint forms the potential between ψ and ϕ). Direct variable elimination does not work, as it would require transmitting a *distribution* over all tracklet-tracklet-group triples. Dual decomposition [42] also results from a Lagrangian formulation, but is different from ours. We employ combinatorial optimization methods inside of max-product (our K-means and Hungarian algorithms) which has been explored in other max-product formulations [16].

Algorithm 1: SGB Algorithm

```
Data: Tracklet set $\tau$
Result: Tracking $\phi_{Final}$, Grouping $\psi_{Final}$
1  for $K \leftarrow 1$ to $K_m$ do
2      for $i \leftarrow 1$ to $N$ do
3          $\mu \leftarrow 0$, $\phi^{K,i} \leftarrow 0$
4          initialize $\psi^{K,i}$ and $G^{K,i}$ randomly
5          while Not local maximum for Eq. 16 do
6              $\mu \leftarrow$ subgradient ascent: Eqs. 17 and 18
7              while $\phi^{K,i}$ or $\psi^{K,i}$ changes do
8                  Update $\phi^{K,i}$: Eq. 19
9                  while $\psi^{K,i}$ changes do
10                     Update $\psi^{K,i}$: Eq. 20
11                     Update $G^{K,i}$ according to $\psi^{K,i}$
12         $Cost^{K,i} \leftarrow$ primal cost ($\phi^{K,i}$, $\psi^{K,i}$, $G^{K,i}$):
           Eq. 14
13 $(K^*, i^*) \leftarrow \arg\min_{K,i} Cost^{K,i} + \beta K$
14 $\phi_{Final} \leftarrow \phi^{K^*,i^*}$, $\psi_{Final} \leftarrow \psi^{K^*,i^*}$
```

5 Socially-aware Head Pose Estim
tion

This section introduces the estimation of head poses gi
grouping information and tracking result ($P(\Upsilon|\phi, \psi, G,$
We formulate this problem as inference in a Conditi
Random Field (CRF), discuss how we build the so
interaction factor, and provide exact convex learning
inference procedures.

5.1 A Conditional Random Field Formulation

$P(\Upsilon|\phi, \psi, G, \tau)$ is the probability of head pose labelin
the video. In this work, we model the head poses of a gr
as a generative graph labeling[1] problem for each grou
each time instance:

$$P(\Upsilon|\phi, \psi, G, \tau) = P(\Upsilon|\phi, \psi, \tau) = \prod_k P(Y_k|X_k, L_k), \tag{21}$$

assuming group mean trajectories do not affect head pose estimation given grouping assignments. Concentrating on a single group (one $P(Y_k|X_k, L_k)$ term), we drop the k subscript. By assuming a uniform prior on head poses, each evidence source is independent given the head pose, and each unary evidence (x) only depends on the person's head pose (y), we have

$$P(Y|X, L) = \frac{1}{Z} P(Y, L) \prod_i \prod_j P(y_j|x_j^i), \tag{22}$$

with Z being a normalization constant. We model pairwise social tendencies in the group for $P(Y, L)$. This problem can be modeled as a CRF as shown in Fig. 3. By using a log-linear model and ignoring the normalization constant, we get:

$$\begin{aligned} \ln P(Y|X, L) &\propto \sum_i \left\langle w_1^i, \Lambda_1^i(X^i, Y)\right\rangle + \langle w_2, \Lambda_2(Y, L)\rangle \\ &= \langle w, \Lambda(X, Y, L)\rangle , \end{aligned} \tag{23}$$

where

$$\Lambda_1^i(X^i, Y) = \sum_j \lambda_1^i(x_j^i, y_j), \tag{24}$$

and

$$\Lambda_2(Y, L) = \sum_{j_1 \prec j_2} \lambda_2(y_{j_1}, y_{j_2}, l_{j_1}, l_{j_2}). \tag{25}$$

$\prec$ is an ordering: we enumerate all unique pairs in a group. The subscript in λ_2 denotes a pairwise term. $\lambda_2(\cdot)$ is the feature vector for a pair of people that jointly models head pose labeling Y and locations L, with details described in Sec. 5.2, w_2 is the weight vector for these features, and $\langle \cdot, \cdot \rangle$ is the dot product. Evidence from unary factors (i.e. λ_1 and Λ_1) is represented similarly. $\Lambda(X, Y, L)$ is the feature vector composed of features from Λ_2 and Λ_1^i for all i (from 1 to M). w is a vector of parameters to be estimated (composed of the weights from w_2 and w_1^i for all i). This formulation allows exact convex learning and inference.

1. We use label and head pose direction interchangeably.

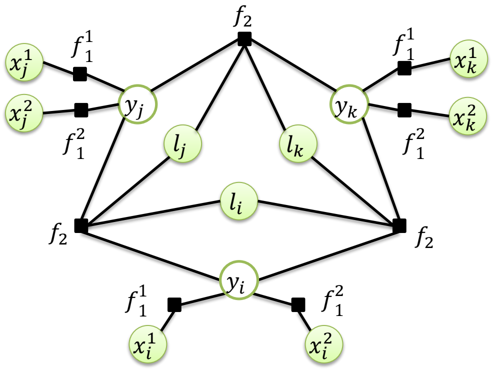


Fig. 3: A factor graph showing how variables and cliques interact in the CRF. A graph of three head images and only two unary features are shown for simplicity. If there are more people in a group or more unary features, this graph can be straight-forwardly augmented.

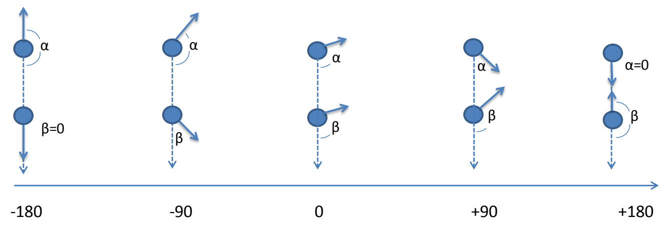


Fig. 4: Structure-aware head pose angle difference. Nodes are head images and dark blue arrows are head directions. Relative positions within group members are considered. The difference is simply $\beta - \alpha$. A positive number implies social attraction.

5.2 Building Group Interaction Models

We study the pairwise head pose interaction patterns in social groups for Eq. 25, which is key for using social grouping information to improve head pose estimation performance. We define the structure-aware head pose angle difference as illustrated in Fig. 4. We will use $SA(j_1, j_2)$, short for $SA(y_{j_1}, y_{j_2}, l_{j_1}, l_{j_2})$, to denote this angle between the head directions of person j_1 and j_2. This angle takes into account the relative positions of the two people. Using structure information allows us to differentiate between social attraction and divergence when the absolute angle difference is the same. Given social groups, we collect such angle differences from only 200 pedestrian pairs from training data (the model data), identify two modes by thresholding velocity (a dataset dependent parameter in pixel/frames similar to that in Chamveha et al. [9]), and build the histograms shown in Fig. 5.

The resulting histograms are intuitive: (1) As shown in Fig. 5 (left), when people walk, they tend to look in the same direction (where they are heading generally or where an object of interest is), but there is more social attraction than divergence, as people tend to make eye contact with each other. We choose to model it with two exponential distributions on both sides of zero degrees. (2) As shown in Fig. 5 (right), when people are relatively stationary, they tend to look directly at each other (angle difference around +180 degree), or be attracted to common objects of interest

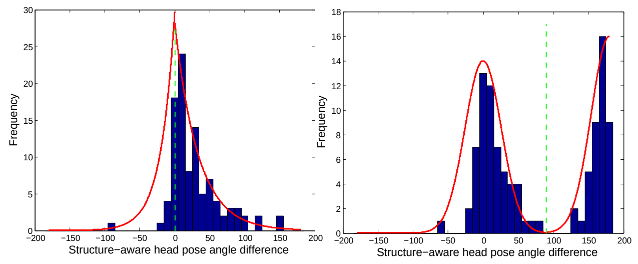


Fig. 5: Two social interaction modes with structure-aware head direction angle difference. Left: dynamical social interaction mode, fitted with two exponentials on either side of 0 degree. Right: static social interaction mode, fitted with two Gaussians on either side of 90 degrees. The specific distributions (exponential and Gaussian) are chosen due to their expressive power in this application and simplicity to express in the negative log space. The fitted distributions are rescaled and are for illustration only; their actual parameters are learned from training data.

(around 0 degrees, for example, when people scan shop windows). Though this is arguably a mixture of Gaussian, we model it with two Gaussians, separating at 90 degrees, for simplicity. The goal of learning is then to learn the rates of the exponentials and variances of Gaussians (feature weights in the negative log space).

These general forms can be converted into features so that the weights in Eq. 25 correspond to the rates and variances above. Given the group head pose interaction models, the feature vector of dynamical interaction mode for two head images (two exponentials on either side of 0 degree) is $\lambda_2^{moving}(y_{j_1}, y_{j_2}, l_{j_1}, l_{j_2}) =$

$$\begin{bmatrix} |SA(j_1,j_2)| \; I[SA(j_1,j_2) \geqslant 0] \\ |SA(j_1,j_2)| \; I[SA(j_1,j_2) < 0] \end{bmatrix}. \tag{26}$$

$I[\cdot]$ is the indicator function indicating the submode of social interaction for the pair $j_1 - j_2$. If the mode is off, the corresponding feature is 0.

The feature vector of the static interaction mode is $\lambda_2^{static}(y_{j_1}, y_{j_2}, l_{j_1}, l_{j_2}) =$

$$\begin{bmatrix} (SA(j_1,j_2) - 180)^2 \; I[SA(j_1,j_2) \geqslant 90] \\ (SA(j_1,j_2))^2 \; I[SA(j_1,j_2) < 90] \end{bmatrix}. \tag{27}$$

Similar to the feature vector in Eq. 26, these two features indicate which Gaussian submode is active and the corresponding feature value.

The dynamical interaction feature and static interaction feature can be unified as $\Lambda_2(y_{j_1}, y_{j_2}, l_{j_1}, l_{j_2}) =$

$$\begin{bmatrix} \lambda_2^{moving}(y_{j_1}, y_{j_2}, l_{j_1}, l_{j_2}) \; I[\text{moving}] \\ \lambda_2^{static}(y_{j_1}, y_{j_2}, l_{j_1}, l_{j_2}) \; I[\text{not moving}] \end{bmatrix}. \tag{28}$$

For example, if people are moving (estimated from tracking result), the dynamical interaction (moving) mode is on, and all features in λ_2^{static} become 0.

5.3 CRF Parameter Learning

Our CRF modeling allows exact convex discriminative learning. Note that we are interested in a regression problem, as the loss function models angle difference. However, using discrete and fine (32 bins) class labels make exact learning possible.

Let $X^{(m)}$ denote all unary features, $L^{(m)}$ denote head locations, and $Y^{(m)}$ denote the ground-truth labeling of group instance m. Further, let $\Lambda^{(m)}(Y) = \Lambda(X^{(m)}, Y, L^{(m)})$; thus $\Lambda^{(m)}(Y^{(m)}) = \Lambda(X^{(m)}, Y^{(m)}, L^{(m)})$ indicates a ground-truth feature-label configuration from training data. We conduct discriminative learning [43] of $P(Y|X, L)$ in the negative log space. Given N training examples, each of which is a graph labeling and related features, the objective function of training is $g(w) =$

$$\frac{1}{N}\sum_{m=1}^{N} \ln \sum_{Y} \left(\frac{P(Y|X^{(m)}, L^{(m)})}{P(Y^{(m)}|X^{(m)}, L^{(m)})} e^{l(Y^{(m)};Y)} \right) + \frac{\gamma}{2}||w||^2, \tag{29}$$

where $l(\cdot;\cdot)$ is the loss function for a group:

$$l(Y^{(m)}; Y) = \sum_j l'(y_j^{(m)}; y_j). \tag{30}$$

$l'(\cdot;\cdot) \in [0, 180]$ is the absolute difference between two directions. $\frac{\gamma}{2}||w||^2$ is a regularization term to avoid overfitting (γ is achieved via cross-validation in training).

After we apply Eq. 23, the objective function becomes

$$g(w) = \frac{1}{N}\sum_{m=1}^{N} \ln \sum_{Y} \Gamma^{(m)}(Y) + \frac{\gamma}{2}||w||^2 \tag{31}$$

where $\Gamma^{(m)}(Y) = e^{l(Y^{(m)};Y) - \left\langle w, \Lambda^{(m)}(Y^{(m)}) - \Lambda^{(m)}(Y) \right\rangle}$, (32)

Eq. 31 is convex with gradient

$$\gamma w - \frac{1}{N}\sum_{k=1}^{N} \frac{\sum_Y \Gamma^{(k)}(Y)(\Lambda^{(k)}(Y^{(k)}) - \Lambda^{(k)}(Y))}{\sum_Y \Gamma^{(k)}(Y)}. \tag{33}$$

Since the objective function and gradient are explicit, minimization can be done exactly with any convex programming package, and we again use the one from Schmidt [39].

The complexity is $O(Q^C)$, where Q is the number of quantized head pose directions and C is the number of people in a group. Sociology research [29] shows that in natural scenes, people generally form groups of fewer than 6 people. This is also validated in the dataset we use. If the scene is really crowded (such as a Marathon event), large groups can be divided into smaller ones, or our model is not suitable since social interaction can be quite noisy in such cases. Running time is discussed in Sec. 7.5.

5.4 Head Pose Estimation Inference

Given model parameters (feature weights learned in the previous section), we perform head pose estimation inference by outputting

$$\max_Y \langle w, \Lambda(X, Y, L) \rangle \; , \tag{34}$$

which is the maximization of the log of $P(Y|X, L)$.

We use a brute-force approach to try all combinations of head directions for exact inference. The complexity is the same as learning, with tractability discussed above.

6 Details on Lower-level Tasks

Our framework is general in that it can be built upon different choices of lower-level components, such as tracklet extraction methods, features to build the tracklet affinity matrix, and unary features used for head pose estimation. We give details of our choices for implementation.

6.1 Parameter Estimation for Tracking

Parameters for tracking and group discovery include the feature weights for tracking, and κ for group number selection. They are estimated by a coarse grid search in the first time window in each dataset, and are fixed afterwards. In practice, feature weights are first selected for tracking without social grouping. Then κ is selected by a simple binary search after adding the social grouping term.

6.2 Tracklet Extraction

Our framework only requires the tracklet extraction method employed to be reliable (commonly assumed in the literature). Namely there should be few within tracklet identity switches. In order to perform comparative experimental evaluation, when tracklets from authors of published work are available, we use them. Otherwise we build our tracklet extraction framework based on human detection responses, combining nearest neighbor association and template matching to extract conservative tracklets. Given detection responses, we link detection response pairs only at consecutive frames which have very similar color, size and position. Additionally, the newly added detection must be similar to the first detection in the tracklet, thus avoiding within-tracklet ID switches caused by gradual changes. We find this simple strategy produces almost zero ID switches within tracklets and good recall performance.

6.3 Basic Affinity Model

Social grouping behavior regularizes the tracking solution and alleviates the need for a highly tuned affinity model. However, the basic affinity model must produce reasonable measurements, $H_{i,j}$. For both single-camera and multi-camera tracking, we build the basic affinity model using appearance (app) cues and spatial-temporal (st, usually referred as motion in single-camera tracking) cues:

$$-\ln P_{link}(i,j) = -\ln p_{i,j}^{app} - \ln p_{i,j}^{st}. \qquad (35)$$

For single-camera tracking, we use the Bhattacharyya distance between the average color histograms within the tracklets [41]. We employ the HSV color space and get a 24-element feature vector after concatenating 8 bins for each channel. The motion model is a simple linear motion smoothness measure [25].

For multi-camera tracking, we use the BTF model and the Parzen window technique for spatial-temporal information in Javed et al. [20]. $P_{init}(T_i)$ and $P_{term}(T_i)$ are set to be a single constant (from training) for simplicity. There is also the time constraint that tracklet linking is only possible when tracklet j takes place later than tracklet i and within a maximum allowed frame gap t_{max}.

6.4 Spatial-temporal K-means Clustering

We describe how to implement the two steps of K-means clustering: group update (with group assignments given) and tracklet assignment (with group parameters given).

Recall that we modeled the group mean trajectory for G_k as, at each time t, a distribution over which camera a member of the group appears in, $b_{k,\cdot}(t)$, and a mean position within each camera a that a group member would appear, $u_{k,a}(t)$. Track assignment (finding ψ given a fixed G) is simple: for each tracklet τ_i, compute $D(\tau_i, G_k)$ from Eq. 11 for each group G_k and select the one that minimizes the negative log-likelihood.

For the update of G_k with the assignment ψ fixed, we must find the parameter assignments to $b_{k,\cdot}$ and $u_{k,\cdot}$ that maximize the likelihood. The log-likelihood is a sum across time, so the maximization can be done independently at each time point. $b_{k,a}(t)$ is a multinomial parameter and therefore its maximum likelihood estimate is proportional to the number of tracklets that are assigned to group k at time t in camera a.

$u_{k,a}(t)$ is the conditional mean for group k at time t in camera a. Therefore, its maximum likelihood parameter is the average position of all tracks assigned to group k at time t in camera a. If at any point there are no tracklets for group k and camera a, we use linear interpolation or extrapolation to generate a mean. If no tracklets in camera a are ever assigned to group k, we place $u_{k,a}(t)$ in the middle of the image for all t.

6.5 Unary Terms in CRF

Features from existing work can be used to construct unary features in our head pose estimation framework. We use two unary features. First, walking direction is shown to be effective in some datasets. As proposed by Benfold and Reid [3] and validated in our work, head pose direction is distributed approximately as a Gaussian with the walking direction as the mean. Thus in our negative log-likelihood framework, the unary feature of walking direction is

$$\lambda_1^{walking}(y_j, x_j^{walking}) = (y_j - x_j^{walking})^2. \qquad (36)$$

We also build a two-level HoG vector to model visual features of head images, following Chen and Odobez [10]. Then we train a multi-class SVM with probability estimates [45]. Besides predicting labels, this allows us to estimate the probability of a visual vector belonging to each class. In this way, we have

$$\lambda_1^{HoG}(y_j, x_j^{HoG}) = -\log P(y_j|x_j^{HoG}). \qquad (37)$$

where $-\log P(y_j|x_j^{HoG})$ can be directly obtained from the output of an SVM classifier with probability estimates.

TABLE 1: Datasets used for each task in the experiments.

Task	Datasets
Multi-target Tracking	PETS 2009, CAVIAR, TUD
Multi-camera Tracking	VideoWeb
Head Pose Estimation	PETS 2009, CAVIAR, TownCentre
Group Discovery	PETS 2009, PSUHub, TownCentre

TABLE 2: Comparison of the tracking result on the CAVIAR dataset: 75 ground truth (GT) tracks.

Method	Recall	Prec.	MT	ML	Frag	IDS
Particle filter	55.7%	60.4%	53.3%	10.7%	15	19
Basic affinity	81.1%	82.7%	77.3%	6.7%	9	12
MCMC[41]	84.5%	90.7%	84.0%	4.0%	6	8
SBM[52]	–	–	85.3%	4.0%	7	7
Our SGB	90.1%	95.1%	88.0%	2.6%	5	6

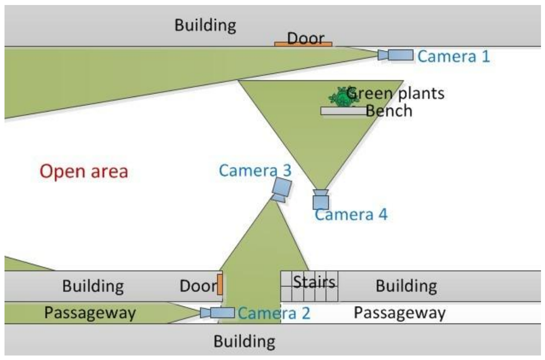


Fig. 7: Topology of the cameras in the experiments.

7 Experiment

We conduct comparative experiments with recent related methods on publicly available datasets for tracking, head pose estimation, and group discovery. Experimental results clearly show the benefits of utilizing social grouping context. The datasets we use is summarized in Tbl. 1

7.1 Single-camera Tracking Evaluation

We first evaluate how modeling social grouping behavior helps to improve single-camera multi-person tracking on the CAVIAR Test Case Scenarios dataset [7]. We use the videos selected by Song et al. [41], consisting of 12,308 frames for about 500 seconds. We retrieve tracklets from the same authors and use the same evaluation metrics by Li et al. [25]: the number of ground truth trajectories (GT), mostly tracked trajectories (MT), mostly lost trajectories (ML), fragments (Frag), ID switches (IDS), and recall and precision for detections. A comparison with several published results under the same configuration is shown in Tbl. 2. Our basic affinity model achieves reasonable results, while better results than competing methods can be achieved by employing our social grouping model with the simple affinity model.

Fig. 6 shows representative cases of the strong grouping information that allows us to improve tracking performance.

We further compare our model on the popular PETS 2009 and TUD-Stadtmitte datasets against a number of state-of-the-art methods using the same evaluation metrics. We obtained the publicly available detection results, ground truth data, and automatic evaluation tool from the authors of [50]. In addition to the former metrics, we also report the false alarm rate (FAF) for detections, and partially tracked trajectory ratio (PT) from the evaluation tool. In Tbl. 3 and Tbl. 4 we can see that our model outperforms several state-of-art methods, even though our model is built upon a simple basic affinity model. On the other hand, competing methods either solve complex optimization problems (Milan et al. [28] introduce six types of jumps in the optimization space) or build sophisticated affinity models (Kuo and Nevatia [21] use appearance features from the person identification literature). Of particular interest, for the PETS 2009 dataset, pedestrians were asked to travel across the scene multiple times. Even in such a scenario they formed groups and made social interactions, which is utilized by our model to help tracking. An example is shown in Fig. 8.

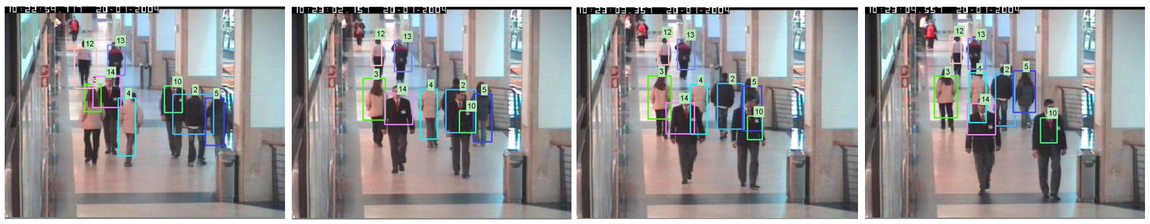
7.2 Multi-camera Tracking Evaluation

We test our method using two sets of videos on the publicly available VideoWeb dataset [14]. We choose Cam27, Cam20, Cam36 and part of Cam21 (indexed by 1–4) to establish the desired non-overlapping topology, shown in Fig. 7. Multi-camera tracking in this setting is very challenging for the following reasons. (1) We use 4 cameras, unlike most prior work that use 2–3. (2) This is an outdoor dataset with a cluttered environment and severe within-camera illumination change, which makes traditional methods that establish one single transformation between each camera pairs, such as BTFs, much less reliable. (3) Since this dataset is mainly designed for complex real-world activity recognition, there exist heavy interactions among individuals, unlike "designed" tracking datasets (for example the one in the work of Javed et el. [20]).

We compare our proposed multi-camera social grouping behavior tracking (MulSGB) to directly using the Bhattacharyya distance between RGB color histograms, Parzen window estimation for spatial-temporal information and the original color histogram for appearance (Parzen Window) and the BTF plus Parzen window estimation framework (Parzen Window + BTF) in the work of Javed et al. [20].

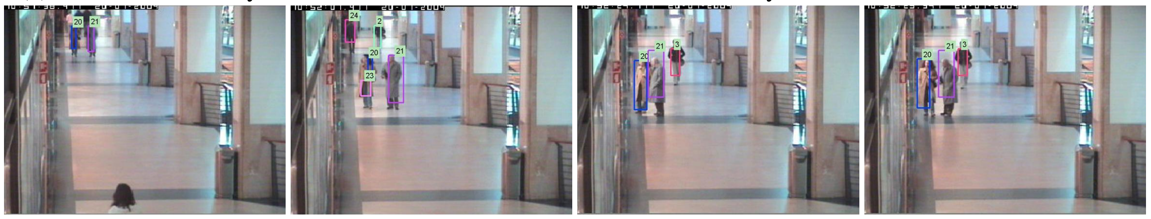
We gather 9 videos using all 4 cameras and 4 videos with camera 1–3. We use 5 videos from the first set for training and all the other videos for testing (note the second set of videos contains a subset of cameras of the first set so no additional training is needed). All other videos in the dataset either had no inter-camera motion or were missing data for more cameras. The data used have roughly 40,000 frames (25fps) for each of the four cameras for training and 80,000 frames for each camera for testing. For detection, we use a state-of-art pedestrian detector [17] to get detection responses and generate reliable intra-camera tracks using our introduced single-camera tracking framework. The same set of tracks are used for all comparing methods. We hand-labeled ground truth and measure the percentage of correctly linked pairs for the eight testing scenes (which consist of 244 single-camera tracks in total). Fig. 9 and Fig. 10 show the results for each set of videos.

We have the following observations. (1) Given the poor color histogram result, especially for the four-camera



Frame 890 Frame 950 Frame 980 Frame 1010

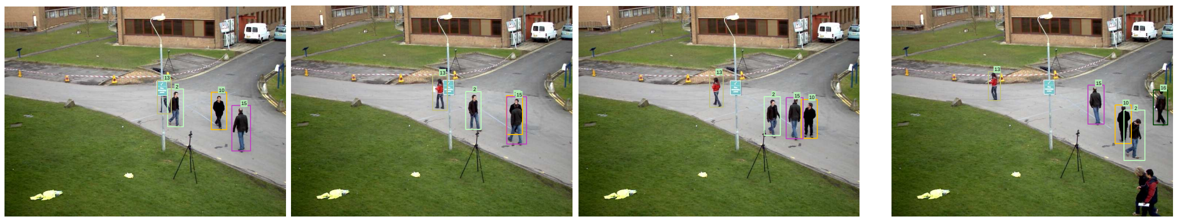
(a) Even under heavy occlusions and interactions, the usual identity switch of 2 and 10 is avoided.



Frame 510 Frame 1245 Frame 1650 Frame 1682

(b) Long-term tracking of the couple (20,21) is possible under challenging conditions: small target, illumination change (frame 510), and false detection (frame 1245).

Fig. 6: Some representative tracking results for CAVIAR dataset.



Frame 477 Frame 483 Frame 489 Frame 509

Though there are heavy interactions between 10 and 15, social context from 2 helps to recover an ID switch.

Fig. 8: One representative tracking result for PETS dataset.

TABLE 3: Comparison of the tracking result on the PETS 2009 dataset.

Method	Recall	Precision	FAF	GT	MT	PT	ML	Frag	IDS
KSP [4]	83.8%	96.3%	0.160	23	73.9%	17.4%	8.7%	22	13
Energy Minimization [28]	92.4%	98.4%	0.070	23	91.3%	4.4%	4.4%	6	11
Online CRF [50]	93.0%	95.3%	0.268	19	89.5%	10.5%	0.0%	13	0
Nonlinear Motion [49]	91.8%	90.0%	0.053	19	89.5%	10.5%	0.0%	9	0
Our SGB model	97.2%	98.6%	0.077	19	94.7%	5.3%	0.0%	4	2

TABLE 4: Comparison of the tracking result on the TUD-Stadtmitte dataset.

Method	Recall	Precision	FAF	GT	MT	PT	ML	Frag	IDS
KSP [4]	63.1%	79.2%	0.650	9	11.1%	77.8%	11.1%	15	5
Energy Minimization [28]	84.7%	86.7%	0.510	9	77.8%	22.2%	0.0%	3	4
PRIMPT [21]	81.0%	99.5%	0.028	10	60.0%	30.0%	10.0%	0	1
Online CRF [50]	87.0%	96.7%	0.184	10	70.0%	30.0%	0.0%	1	0
Our SGB model	95.2%	98.5%	0.085	10	90.0%	10.0%	0.0%	4	3

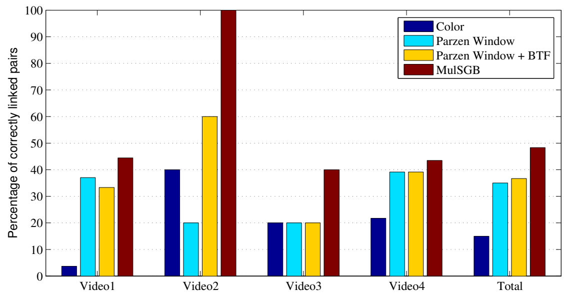


Fig. 9: Percentage of correctly linked pairs on the four video sequences with four cameras. The videos consist of 27, 5, 5 and 23 (60 in total) ground truth linked pairs respectively.

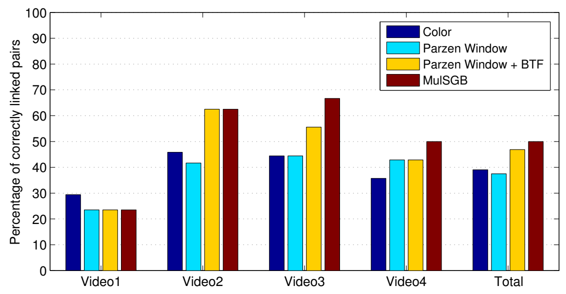
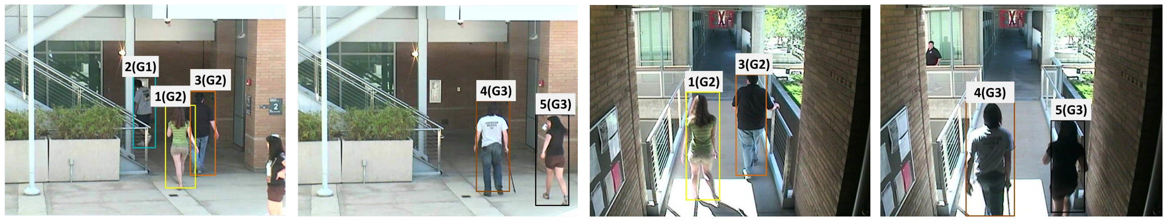


Fig. 10: Percentage of correctly linked pairs on the four video sequences with three cameras. The videos consist of 17, 24, 9 and 14 (64 in total) ground truth linked pairs respectively.



Frame 5809 (Cam 3) Frame 6222 (Cam 3) Frame 6295 (Cam 2) Frame 6689 (Cam 2)

Fig. 11: Example tracking result with our model, where G indicates group number. Because people form groups and show proximity to group members, social grouping provides powerful contextual information to improve multi-camera tracking. Other methods tend to identify a new person (Frame 6295 target 1) and output an identity switch (target 3 and 5) on this sequence, because traditional evidences are unreliable.

setting (demonstrating the difficulty of the dataset), the overall performance is good, as our MulSGB model indeed improves tracking performance over competing methods. (2) The example in Fig. 11 shows a representative example where social grouping helps tracking, while other methods fail under this challenging sequence. (3) Since our social grouping model serves as a regularizer, the basic affinity model upon which we built social grouping model is sometimes a bottleneck. For example, we observe no improvement upon the baseline model for two sequences in Fig. 10. We observed that in such cases, although the optimization usually heads toward a good solution, it could not recover wrong links since the basic model provides very unlikely handover possibility between the correct pairs. For example, when the illumination condition changes between the testing set and training set, the learned BTF may even hurt the performance comparing to pure color histogram comparison, as is the case for video1 in Fig. 10.

7.3 Head Pose Estimation Evaluation

We evaluate how social interaction improves head pose estimation in challenging videos, using the TownCentre dataset [3], CAVIAR, and PETS 2009. We use mean absolute angle difference (MAAD) stated in degrees as the evaluation metric, as is commonly done in related work. We quantized head pose into 32 directions, which is finer than most existing work (such as 8 directions [10][38]). This helps alleviating errors from coarse quantization when comparing angles. Competing methods that require discretization use the same setting.

We compare our method with models using visual features only (HoGSVM) and walking direction only (Walking). We also compare our method with a model with both visual and motion features. We call this model the BR (Benfold and Reid) setting [3]. Our implemented BR baseline does not incorporate temporal information. However, the resulting CRF can be solved exactly. We feel these two factors largely compensate each other as we get comparable results as those by Benfold and Reid [3]. Temporal information might be incorporated in our framework if approximate inference algorithms were applied. We also compare with two state-of-the-art methods: Orozco et al. [31] build a mean image for each class and represent each image as a distance map to these references. We use our own implementation with KL-Divergence as the distance

TABLE 5: Comparison of the head pose estimation results on the TownCentre, CAVIAR and PETS 2009 dataset. Numbers are reported on MAAD.

Method	TownCentre	CAVIAR	PETS
HoGSVM	31.20	28.80	32.64
Walking	23.89	72.01	58.28
DisMap [31]	33.12	30.20	31.54
WARCO [44]	31.12	25.70	28.65
BR Setting [3]	22.87	27.00	31.85
Ours	21.83	24.65	28.78

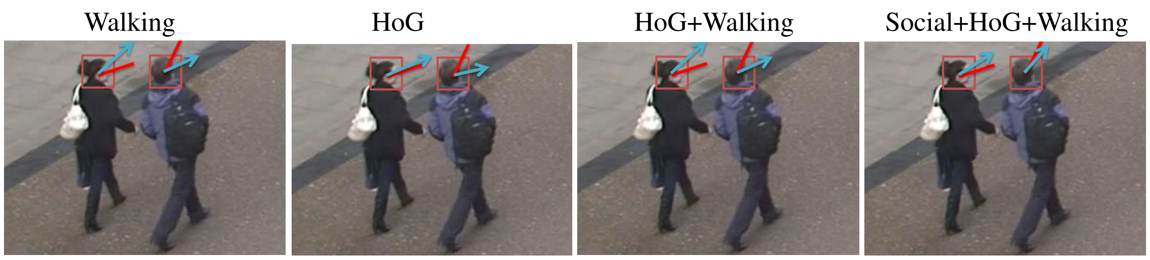
measure (best reported measure in the paper). Tosato et al. [44] design a new visual feature and have publicly available implementation. Note that the small-sized head images make the comparison to landmark detection based work (e.g. [53]) impossible.

We use head images from people that are not in groups to train the multi-class SVM. Note we only report results for people identified in groups. For people that are not identified in groups, our model would output exactly the same result by using individual features alone. For the TownCentre dataset, about 30% of the people are identified in groups. For PETS 2009, over 40% of the people are in social groups. For the CAVIAR dataset over 60% are in groups.

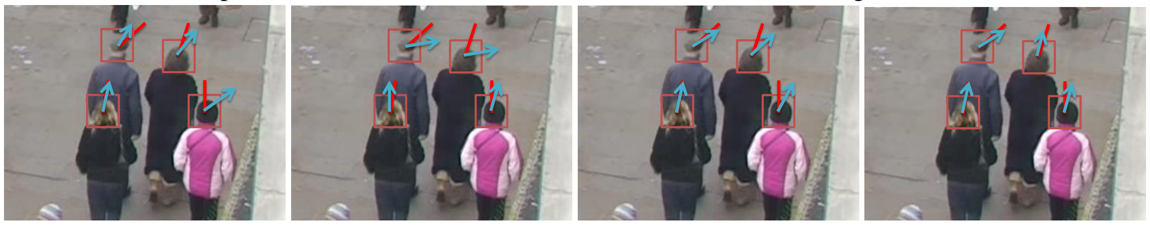
We first use the TownCentre dataset to test our proposed method. This dataset has been used in several recent papers. It involves people traveling in a shopping mall. Though this dataset is treated as high-resolution video in the tracking literature, head images are small due to the high camera angle. We use the result of head tracking from Benfold and Reid [3] and use our spatial-temporal clustering procedure in Sec. 6.4 to determine groups. We manually label head directions for every 15 frames. Due to annotation differences, the angle differences are not directly comparable. But the performance we get from our BR setting baseline implementation is comparable to that of Benfold and Reid [3], which reports an MAAD of 23.90.

We gather 270 pairs of head images for this dataset. Whenever training is involved, 100 pairs are used for training and the others are used for testing. Since camera parameters are available for this dataset, we evaluate performance on the ground-plane. The results for different methods are shown in Tbl. 5.

As stated by Benfold and Reid [3], we also observe that walking direction provides a very good baseline in this dataset since most people are walking in the shopping

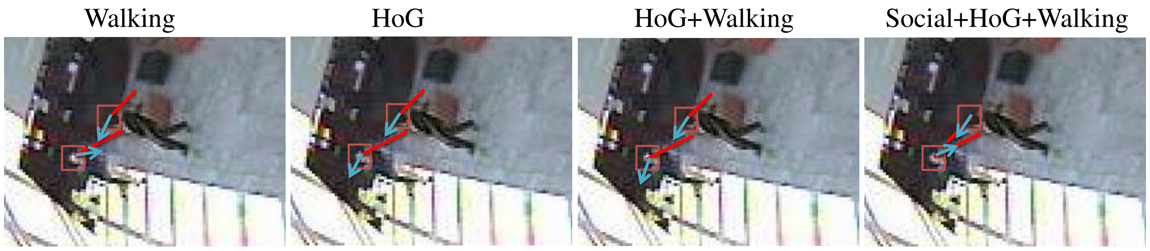


(a) Our model provides finer head direction estimate even when walking direction is reliable.

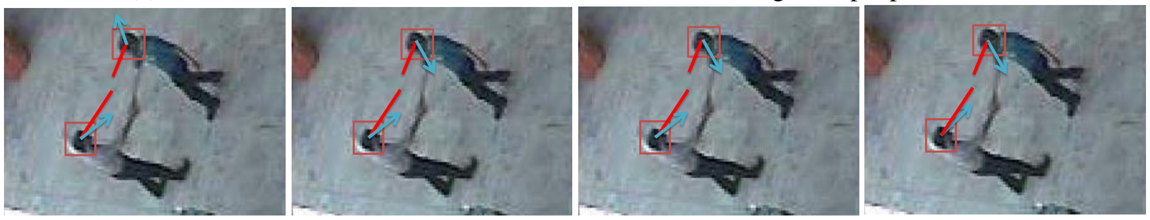


(b) Our model helps to correct head direction estimations in social groups with multiple people.

Fig. 12: Representative head direction estimation results for TownCentre. Red lines indicate human-labeled head direction.



(a) Our model corrects head direction for small head images as people interact.



(b) One case that our model is not able to fully recover false estimations.

Fig. 13: Some representative head direction estimation result for CAVIAR dataset.

mall. It can generate a better result than using only visual features. But even in such scenario, our model improves upon the best non-social method. As people walk together, their head directions tend to be attracted by other group members. Using social information regularizes out outliers that do not conform to such social constraints. Note that the performance gain from our social model is as large as the gain from combining two non-social information sources (comparing to using walking direction alone). We show two qualitative examples in Fig. 12.

We also compare performances on the CAVIAR dataset and PETS 2009 dataset. We annotate 5 video sequences[2] in CAVIAR and the entire PETS dataset at every 5 frames for head locations and head direction manually to focus on head pose estimation. For CAVIAR we gather 241 pairs of data, 100 of which are used for training and the others for testing. For PETS we gather 194 pairs of data and use half of them for training. Note for these two datasets we directly assign person ID and group ID based on our tracking model. That is, we do not assume ground truth identity or group member labeling and we evaluate head pose estimation performance in the complete system.

Compared to the TownCentre dataset, head images in these two datasets are of lower resolutions but possess lower variance because there are fewer people. CAVIAR involves more people standing still; the static mode of our social interaction model is more frequently activated and walking directions can be very noisy. People in PETS also show more freedom while walking so walking direction is again not as reliable as that in TownCentre. For these two datasets, we evaluate performance on the image plane.

We summarize the results in Tbl. 5. The performance gains by incorporating social context are more significant on these two datasets. They are much larger than the gain from combining the two non-social information sources (comparing to using visual feature alone.) This is because walking direction is often no longer a reliable feature and visual features are still weak. Yet, when people are relatively static, they tend to make more social contacts so our model helps more. Also, when walking, pedestrians' head direction severely deviates from walking direction,

2. FightChase, MeetSplit3rdGuy, FightOneManDown, MeetWalkTogether1, FightRunAway1

such deviations are usually motivated by group members or objects of interest, which is modeled in our formulation. We note that the reference-set based approach [31] does not perform very well due to its classification (instead of regression) formulation and the sparsity of training data. Our model performs comparatively with or better than the state-of-the-art method [44]. Some examples are shown in Fig. 13. We also show a case where our social model is not able to recover from false head pose estimations: Fig. 13(b). This is because our social model can be viewed as a regularizer, and it will not help much when the baseline model provides very bad evidence (for example, assigning very low probability to the true label).

7.4 Group Discovery Evaluation

Group discovery is provided by the group assignment matrix of our model. The simple spatial-temporal clustering approach is robust as a global consistency measure, while existing methods typically use features such as velocity, which can be unreliable with noisy detections or standing-still people. We show that our group discovery component can produce reasonable result compared to other designed approaches. The fact that our group discovery model is coupled with the tracking process (while other methods typically assume and are built upon perfect tracking result) makes our grouping approach more practical. When trajectories are available, the spatial-temporal clustering approach can be directly applied. We evaluate both cases.

Following Ge et al. [18], we use the following evaluation method: Each pedestrian is coded into one of two categories: alone or in a group. This is called the dichotomous coding scheme. A trichotomous coding scheme classifies each pedestrian into alone, in a group of two, or in a group of three or more. Match rate indicates the percentage of persons that are classified correctly. Furthermore, to test the statistical significance of the agreement between the human annotations and the output of the algorithm, Cohen's Kappa test [24] is used. Kappa score ranges from -1 to 1, and Landis and Koch [24] characterize values smaller than 0 as indicating no agreement and $(0, 0.2]$ as slight, $(0.2, 0.4]$ as fair, $(0.4, 0.6]$ as moderate, $(0.6, 0.8]$ as substantial, and $(0.8, 1]$ as almost perfect agreement.

Since we are not aware of group discovery results or annotations on the datasets we conduct tracking experiments on, or any available implementations of relating work, we are not able to conduct comparative experiments on these datasets. We thus annotate grouping in the PETS 2009 dataset. Our method produces 87% matching rate and a κ value of 0.75 for both dichotomous and trichotomous coding scheme (there are no trichotomous groups in the ground truth.) 55 trajectories are identified in time windows of 100 frames. (The same person in different time windows are treated as different persons [18].) We can achieve substantial agreement with human annotator on this dataset. If we focus on predicted pairs of people in social groups, for the 11 groundtruth pairs, our system achieves 91% recall and 71% precision.

TABLE 6: Comparison of the group discovery result on the PSUHub dataset.

	Match Rate	κ
dichotomous [18]	84%	0.74
trichotomous [18]	75%	0.63
dichotomous [ours]	83%	0.58
trichotomous [ours]	76%	0.49

We also compare our method with Chamveha et al [8] on the Towncentre dataset. Since their implementation is not available, we report the same measure, group accuracy (whether two people are in a group or not, compared with human annotation), as reported in the paper on the same dataset. We achieve an accuracy of 78.2% while they report 81.8%. The results are comparable and their method is based on the ground truth trajectories.

We further test our spatial-temporal clustering method against Ge et al. [18] on their publicly available PSUHub dataset and compare with their results. The dataset provides 2476 pedestrian trajectories in 177 time windows without images. We show the results in Tbl. 6.

We achieve comparative matching rates to a method designed solely for group discovery. Our model is inferior in terms of Kappa test, but we still get moderate agreement with ground truth. Note that our model is very simple to implement with only one parameter (weight for group size penalization, which is fixed across each dataset), while we are aware of at least four free parameters in Ge et al. [18]. Also, our method tends to group strangers that follow common path. Such pragmatic social groups still help tracking and head pose estimation. (Strangers may still follow common path, look at where they are heading to, or look at common object of interest.) Furthermore, the coupling of our clustering method with tracking makes it more practical when full trajectories are not available.

7.5 Running Time

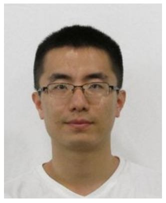
We use a standard desktop and all our code is implemented in Matlab without specific optimization or parallelization. For the tracking problem, given tracklets and the affinity matrix H, the running time of our optimization depends on the implementation of the second-order gradient based method and scales with the number of tracklets. For the datasets in this paper, it takes 1 to 10 seconds to converge to a local maximum for each run on a time window. Though multiple runs with different random initializations are necessary to find a better optimum, our optimization is trivial to parallelize for each run. For head pose estimation, our implementation for training takes about one minute to converge to the global optimum with 100 pairs of data. Testing typically takes fewer than 5 seconds to finish (since no gradient descent is involved). Group discovery given full trajectories takes less than one second for each time window for the PSUHub dataset.

8 Conclusion

We show a general framework of coupling the novel social grouping context with important computer vision tasks including multi-target tracking and head pose estimation. Certain sub-components in our framework are naturally coupled and thus can be joint optimized. We then provide effective solvers for those components based on nonlinear optimization and conditional random field. We conduct extensive experiments to show that social grouping context helps tracking and head pose estimation. Our social grouping model alone can also produce reasonable results.

REFERENCES

[1] J. Aghajanian and S. Prince. Face pose estimation in uncontrolled environments. In *BMVC*, 2009.
[2] L. Bazzani, M. Cristani, and V. Murino. Decentralized particle filter for joint individual-group tracking. In *CVPR*, 2012.
[3] B. Benfold and I. Reid. Unsupervised learning of a scene-specific coarse gaze estimator. In *ICCV*, 2011.
[4] J. Berclaz, F. Fleuret, E. Türetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Trans. PAMI*, 2011.
[5] W. Brendel, M. Amer, and S. Todorovic. Multiobject tracking as maximum weight independent set. In *CVPR*, 2011.
[6] A. A. Butt and R. T. Collins. Multi-target tracking by Lagrangian relaxation to min-cost network flow. In *CVPR*, 2013.
[7] CAVIAR. Caviar dataset. http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.
[8] I. Chamveha, Y. Sugano, Y. Sato, and A. Sugimoto. Social group discovery from surveillance videos: A data-driven approach with attention-based cues. In *BMVC*, 2013.
[9] I. Chamveha, Y. Sugano, D. Sugimura, T. Siriteerakul, T. Okabe, Y. Sato, and A. Sugimoto. Head direction estimation from low resolution images with scene adaptation. *CVIU*, 2013.
[10] C. Chen and J. Odobez. We are not contortionists: Coupled adaptive learning for head and body orientation estimation in surveillance video. In *CVPR*, 2012.
[11] X. Chen, Z. Qin, L. An, and B. Bhanu. An online learned elementary grouping model for multi-target tracking. In *CVPR*, 2014.
[12] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.
[13] M. Demirkus, J. Clark, and T. Arbel. Robust semi-automatic head pose labeling for real-world face video sequences. *Multimedia Tools and Applications*, 2014.
[14] G. Denina, B. Bhanu, H. Nguyen, C. Ding, A. Kamal, C. Ravishankar, A. Roy-Chowdhury, A. Ivers, and B. Varda. Videoweb dataset for multi-camera activities and non-verbal communication. *Distributed Video Sensor Networks*, 2010.
[15] T. D'Orazio, P. Mazzeo, and P. Spagnolo. Color brightness transfer function evaluation for non-overlapping multi-camera tracking. In *ICDSC*, 2009.
[16] J. Duchi, D. Tarlow, G. Elidan, and D. Koller. Using combinatorial optimization within max-product belief propagation. In *NIPS*, 2006.
[17] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Trans. PAMI*, 2010.
[18] W. Ge, R. Collins, and C. Ruback. Vision-based analysis of small groups in pedestrian crowds. *IEEE Trans. PAMI*, 2011.
[19] J. F. Henriques, R. Caseiro, and J. Batista. Globally optimal solution to multi-object tracking with merged measurements. In *ICCV*, 2011.
[20] O. Javed, K. Shafique, Z. Rasheed, and M. Shah. Modeling inter-camera space-time and appearance relationships for tracking across non overlapping views. In *CVIU*, 2008.
[21] C.-H. Kuo, , and R. Nevatia. How does person identity recognition help multi-person tracking? In *CVPR*, 2011.
[22] C.-H. Kuo, C. Huang, and R. Nevatia. Inter-camera association of multi-target tracks by on-line learned appearance affinity models. In *ECCV*, 2010.
[23] C.-H. Kuo, C. Huang, and R. Nevatia. Multi-target tracking by on-line learned discriminative appearance models. In *CVPR*, 2010.
[24] J. Landis and G. Koch. The measurement of observer agreement for categorical data. In *Biometrics*, 1977.
[25] Y. Li, C. Huang, and R. Nevatia. Learning to associate: Hybrid-boosted multi-target tracker for crowded scene. In *CVPR*, 2009.
[26] D. Makris, T. Ellis, and J. Black. Bridging the gaps between cameras. In *CVPR*, 2004.
[27] M. Marin-Jimenez, A. Zisserman, M. Eichner, and V. Ferrari. Detecting people looking at each other in videos. In *IJCV*, 2014.
[28] A. Milan, S. Roth, and K. Schindler. Continuous energy minimization for multitarget tracking. *IEEE Trans. PAMI*, 2014.
[29] M. Moussaid, N. Perozo, S. Garnier, D. Helbing, and G. Theraulaz. The walking behavior of pedestrian social groups and its impact on crowd dynamics. *PLoS ONE*, 5, 2010.
[30] E. Murphy-Chutorian and M. Trivedi. Head pose estimation in computer vision: A survey. *IEEE Trans. PAMI*, 31(4):607–626, 2009.
[31] J. Orozco, S. Gong, and T. Xiang. Head pose classification in crowded scenes. In *BMVC*, 2009.
[32] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *ICCV*, 2009.
[33] S. Pellegrini, A. Ess, and L. Van Gool. Improving data association by joint modeling of pedestrian trajectories and groupings. In *ECCV*, 2010.
[34] H. Pirsiavash, D. Ramanan, and C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR*, 2011.
[35] B. Prosser, S. Gong, and T. Xiang. Multi-camera matching using bi-directional cumulative brightness transfer functions. In *BMVC*, 2008.
[36] Z. Qin and C. R. Shelton. Improving multi-target tracking via social grouping. In *CVPR*, 2012.
[37] Z. Qin, C. R. Shelton, and L. Chai. Social grouping for target handover in multi-view video. In *ICME*, 2013.
[38] N. Robertson and I. Reid. Estimating gaze direction from low-resolution faces in video. In *ECCV*, 2006.
[39] M. Schmidt. minfunc. http://www.di.ens.fr/~mschmidt/Software/minFunc.html.
[40] J. Sochman and D. Hogg. Who knows who - inverting the social force model for finding groups. In *ICCV Wrkshps*, 2011.
[41] B. Song, T.-Y. Jeng, E. Staudt, and A. K. Roy-Chowdhury. A stochastic graph evolution framework for robust multi-target tracking. In *ECCV*, 2010.
[42] D. Sontag, A. Globerson, and T. Jaakkolar. Introduction to dual decomposition for inference. In *Optimization for Machine Learning*, 2011.
[43] C. Sutton and A. McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 2012.
[44] D. Tosato, M. Spera, M. Cristani, and V. Murino. Characterizing humans on riemannian manifolds. *IEEE Trans. PAMI*, 2013.
[45] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res.*, 2004.
[46] Z. Wu, T. H.Kunz, and M. Betke. Efficient track linking methods for track graphs using network-flow and set-cover techniques. In *CVPR*, 2011.
[47] K. Yamaguchi, A. C. Berg, T. Berg, and L. Ortiz. Who are you with and where are you going? In *CVPR*, 2011.
[48] B. Yang, C. Huang, and R. Nevatia. Learning affinities and dependencies for multi-target tracking using a CRF model. In *CVPR*, 2011.
[49] B. Yang and R. Nevatia. Multi-target tracking by online learning of non-linear motion patterns and robust appearance models. In *CVPR*, 2012.
[50] B. Yang and R. Nevatia. Multi-target tracking by online learning a crf model of appearance and motion patterns. In *IJCV*, 2014.
[51] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 2006.
[52] S. Zhang, A. Das, C. Ding, and A. Roy-Chowdhury. Online social behavior modeling for multi-target tracking. In *CVPR Wrkshps*, 2013.
[53] X. Zhu and D. Ramanan. Face detection, pose estimation and landmark localization in the wild. In *CVPR*, 2012.



Zhen Qin Zhen Qin recieved his B.E. degree from Beijing University of Posts and Telecommunications in 2010 and Ph.D. from University of California, Riverside 2015. His research interests are computer vision and machine learning.His research interests are computer vision and machine learning.His research interests are computer vision and machine learning.

Christian R. Shelton Christian R. Shelton received his B.S. degree from Stanford in 1996 and Ph.D. from MIT 2001. After begin a postdoctoral scholar at Stanford, he joined the faculty at the University of California at Riverside where he is currently an Associate Professor of Computer Science. His research interests are in statistical approaches to artificial intelligence.

Appendix F

Marked Point Process for Severity of Illness Assessment

Appeared as

Kazi T. Islam, Christian R. Shelton, Juan I. Casse, and Randall Wetzel. Marked Point Process for Severity of Illness Assessment In *Proceedings of Machine Learning for Healthcare*, 2017.

Marked Point Process for Severity of Illness Assessment

Kazi T. Islam KAZI.ISLAM@EMAIL.UCR.EDU
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California 92507, USA

Christian R. Shelton CSHELTON@CS.UCR.EDU
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California 92507, USA

Juan I. Casse JCASSE@CS.UCR.EDU
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California 92507, USA

Randall Wetzel RWETZEL@CHLA.USC.EDU
Laura P. and Leland K. Whittier Virtual Pediatric Intensive Care Unit
Childrens Hospital LA
Los Angeles, CA 90089, USA

Abstract

Electronic Health Records (EHRs) consist of sparse, noisy, incomplete, heterogeneous and unevenly sampled clinical data of patients. They include physiological signals, lab test results, procedural events, clinical notes. Such data can be treated as a temporal stream of events of varied types occurring at irregularly spaced time points. We focus on modeling the temporal dependencies that arise due to the types, timings, and values of different events in such data. We model the event streams, including vital signs, laboratory results contained in two different datasets (MIMIC III — Medical Information Mart for Intensive Care clinical database — and data extracted from EHRs of patients in a tertiary pediatric intensive care unit) using a piecewise-constant conditional intensity model (PCIM), a type of marked point process. Our experiments capture meaningful temporal dependencies and show improvement in hospital mortality prediction over traditional ICU scoring systems.

1. Introduction

Electronic Health Records (EHRs) contain detailed records of patients symptoms, demographics, outcomes, and other encounter information. In this paper, we concentrate on records from stays in intensive care units (ICUs), because they provide an intense, well monitored, and (relatively) short duration episodes. These attributes allow for the collection of large number of patient trajectories with definite outcomes. We expect many of the modeling lessons learned in this setting to be transferable to other areas of patients' medical records.

We view the EHR data of a patient as a timeline of events happening throughout the patient's stay in the ICU, where an event for an individual patient is a new measurement of a particular physiological variable, a new lab test, a dosage of a particular drug, or a procedure performed, started, or stopped. The measurements associated with such events are indicative of patient states and the subject of constant monitoring from clinicians. But, the temporal dynamics of the stream of events carry extra information for event forecasting and understanding severity of illness. For example, abnormal values for blood pressure could indicate critical states of a patient, but a higher measurement *rate* could also indicate high degree of urgency. Higher values of a certain physiological variable could instigate the clinicians to order a particular lab test. High values of that lab result could result in ordering of a related lab result, or initiate the dosage of a particular drug. These complex temporal dependencies that arise due to time, value, and types of different events throughout the timeline of a patient's stay in an ICU could be vital in understanding the temporal dynamics of the patient's state(e.g. severity of illness). These sort of data are not missing-at-random (or, more accurately, "measured-at-random").

One challenge in modeling medical data is the irregular arrival of different events. Not only are the temporal patterns of measurements indicative of the patients' conditions, but those patterns do not correspond to regular sampling intervals and are dependent on the previous history. Therefore, in this paper we model the timing information in the continuous-time domain. As a result, no fixed sampling rate had to be chosen. Also, the heterogeneous nature of the data requires a model that can capture temporal dependencies between events of varied types. These properties of the complex EHR data stream has led to our choice of using a non-Markovian marked point process model named PCIM (Gunawardana et al., 2011).

Our general goal is to explore how modeling the temporal dynamics of the raw EHR data stream in continuous time domain could facilitate certain critical decision making tasks performed in an ICU. To achieve that goal, we have performed mortality prediction in the ICU using the first 24 hours of ICU data from two different datasets, including the publicly available MIMIC-III database (Johnson et al., 2016).

Severity of Illness (SOI) assessment and mortality modeling is a broad area of research in health informatics. SOI and mortality scores are used to predict patient outcome and often used as the principal measures of quality-of-care comparison among ICUs and stratification for clinical trials. Scoring systems like SAPS-I (Simplified Acute Physiology Score) (Le Gall et al., 1984), SAPS-II (Le Gall et al., 1993), PIM2 (Pediatric Index of Mortality) (Slater et al., 2003), and PRISM3 (Pollack et al., 1996) are the accepted current practices in ICU acuity scoring, but are based on static snapshots of certain clinical variables over a patient's stay in the ICU and ignore the rapidly evolving temporal dynamics of the clinical variables. Several works in the literature have focused on boosting the predictive power of such scoring systems using rich clinical information present in an EHR such as, clinical notes (Ghassemi et al., 2014; Lehman et al., 2012). While the temporal dynamics of the topics derived from the notes using traditional topic modeling techniques are useful (Jo et al., 2015; Ghassemi et al., 2015) for patient risk assessment, the temporal information present in patient trajectories of raw vital signs or lab measurements has not been extensively studied. We conclude that temporal modeling of clinical variables including vital signs and lab tests can complement the standard scoring systems and improve mortality prediction performance.

In this paper we make the following contributions.

- We model the temporal dependency in streams of measurement data using piecewise-constant conditional intensity models (PCIMs).
- For modeling the measurement values we extend the PCIM model to allow marks that contain continuous-values, in addition to the discrete-valued labels.
- From the learned models, we assign risk scores to each patient trajectory with vital signs and lab tests and use these as inputs to a feedforward neural network (Hornik et al., 1989) to predict mortality.

2. Methods

As the data are irregularly sampled, we modeled the timings of the measurements in the continuous-time domain. We discuss the marked point process models used in the next subsection.

Outside of health care, temporal event sequences has been studied in other areas including genetics (Friedman et al., 2000), data center management (Oliner and Stearley, 2007), neuroscience (Truccolo et al., 2005), and web search query logs. Event streams can be modeled in either discrete or continuous time. Discrete time approaches, such as Hidden Markov Models (HMMs) (Rabiner, 1989; Leonard E. Baum, 1966) and Dynamic Bayesian Networks (DBNs) (Dean and Kanazawa, 1988), require discretizing event times to a fixed sampling rate. The irregular sampling property of the EHR data makes the discrete approach less appealing. Recent approaches in modeling continuous time processes include Continuous Time Bayesian Networks (CTBNs) (Nodelman et al., 2002), Continuous Time Noisy-Or (CT-NOR) (Simma et al., 2012), Poisson Cascades (Simma and Jordan, 2012), Poisson Networks (Rajaram et al., 2005; Truccolo et al., 2005), and piecewise-constant conditional intensity models (PCIMs) (Gunawardana et al., 2011). We chose the last, PCIM, which is a class of marked point process, due to its flexible representation of the rate function as a decision tree. This promotes an interpretable and concise representation of the temporal dependencies.

2.1 Piecewise-Constant Conditional Intensity Model

Assume events are drawn from a finite label set L, representing the different event types. An event can then be represented by a pair: a time stamp t and a label $l \in L$. An event sequence x is $\{(t_i, l_i)\}_{i=1}^{n}$, where $0 < t_1 < \ldots < t_n$. We use $h_i = \{(t_j, l_j) | (t_j, l_j) \in x, t_j < t_i)\}$ for the history of event i. Let $t(y)$ for an event sequence y be the time of the last event in y, such that $t(h_i) = t_{i-1}$.

A conditional intensity function $\lambda(t|x)$ associated with a temporal point process is the expected instantaneous rate at which events are expected to occur at time t given the history before t. A conditional intensity model (CIM) is a set of such non-negative conditional intensity functions indexed by the labels $\{\lambda_l(t|x,\theta)\}_{l \in L}$. The likelihood of event sequence x can then be written as

$$p(x|\theta) = \prod_{l \in L} \prod_{i=1}^{n} \lambda_l(t_i|h_i;\theta)^{\mathbf{1}_l(l_i)} e^{-\Lambda_l(t_i|h_i;\theta)}, \tag{1}$$

where $\Lambda_l(t|h;\theta) = \int_{t(h)}^{t} \lambda_l(\tau|h;\theta)d\tau$. If $l' = l$, the indicator function $\mathbf{1}_l(l')$ is one, and it is zero otherwise. $\lambda_l(t|h;\theta)$ is the expected rate of event l at time t given h and model parameters θ. As it is conditional on the entire history, the process is non-Markovian.

A PCIM is a class of CIM where the conditional intensity function is a piecewise-constant function of time for any history. For each label l, a local structure S_l specifies regions in the timeline, where the conditional intensity function is constant and local parameters for each label θ_l represent the values of the intensity function in those regions. Formally, PCIMs are composed of local structures $S_l = (\Sigma_l, \sigma_l(t,x))$ and local parameters $\theta_l = \{\lambda_{ls}\}_{s\in\Sigma_l}$, where Σ_l denotes the set of states where the conditional intensity function is constant, λ_{ls} are non-negative constants representing the intensities in those states, and σ_l is a piecewise constant state function in time that maps a time and a history to Σ_l. In a PCIM, the state function for each label, σ_l, is represented using a decision tree where the states $s \in \Sigma_l$ are the leaves and the internal nodes are binary test functions, formally defined as basis state functions (Gunawardana et al., 2011). They map a time t and a history h to a subtree. If the test functions are picked to be piecewise-constant functions of time for any event history, the intensity function $\lambda_l(t|h) = \lambda_{ls}$, where $s = \sigma_l(t,h)$ becomes piecewise-constant as well. The resulting likelihood of the event sequence x can then be written as

$$p(x|S,\theta) = \prod_{l\in L}\prod_{s\in\Sigma_l} \lambda_{ls}^{c_{ls}(x)} e^{-\lambda_{ls}d_{ls}(x)} \ , \tag{2}$$

where $S = \{S_l\}_{l\in L}, \theta = \{\theta_l\}_{l\in L}$. c and d are the sufficient statistics for likelihood calculation. $c_{ls}(x)$ is the total number of events of label l occurring in x that map to state s, and $d_{ls}(x)$ is the total duration when the trajectory for l is mapped to s. An example of a PCIM model is given in Figure 1.

The basis state functions or the test functions need to be carefully chosen to control the capacity of the resulting model. One of the approaches is to index the functions based on predefined time windows and thresholds. Some examples are

- Is the time of day between 6am and 9am?
- Is the number of events with the label B in the past half an hour greater than a threshold?
- Was the most recent event of label A?

The piecewise-constant assumption allow for efficient inference and learning of the model. Gunawardana et al. (2011) showed that the marginal likelihood of an event sequence, x, can be computed in closed form given the structure S using a product of gamma distributions as a conjugate prior for θ. For learning the decision trees greedily, imposing a structural prior allows a closed form Bayesian score to be computed. Given a structure, the rates associated with the states (leaves) can be selected using maximum a posteriori or maximum likelihood estimation.

2.2 Extending PCIM

In the previous sections, we have represented an event sequence as a sequence of pairs, where each pair consists of the time stamp and label representing the time and type of a particular event. For EHR data, we take the label to be the type of measurement (pulse, for example). However, these events also have associated values (the heart rate measurement,

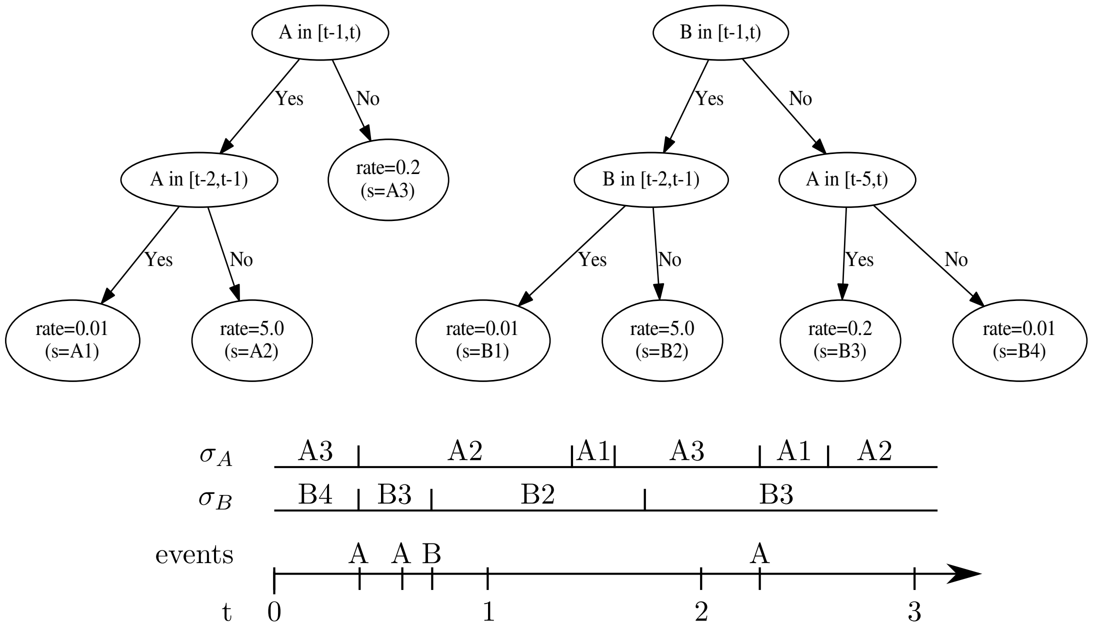


Figure 1: Top: Example decision trees representing a PCIM for two labels A and B. From Gunawardana et al. (2011). Bottom: Example trajectory and the corresponding piecewise-constant assignment of time to states (leaves of the PCIMs).

for example). In this paper, we have focused only on events with numerical values. However, we expect our modeling lessons to be transferable to other forms of event data including clinical notes which can be represented in numerical format, such as topic proportions.

Incorporating the values, an event is now a triple: a time stamp t, a label $l \in L$ and the value of the event $v \in \mathcal{R}$. The notation for an event sequence x then becomes $\{(t_i, l_i, v_i)\}_{i=1}^{n}$, where $0 < t_1 < \ldots < t_n$. We have extended the original PCIM model to model the values in each state $s \in \Sigma_l$. We impose a Gaussian distribution $\frac{1}{\sigma_{ls}\sqrt{2\pi}} e^{-\frac{1}{2\sigma_{ls}^2}(v-\mu_{ls})^2}$ on the value v associated with an event from a particular state s, represented by a leaf in the decision tree. The parameter set θ_l is now $\{\lambda_{ls}, \mu_{ls}, \sigma_{ls}\}_{s\in\Sigma_l}$, and, after incorporating the product of Gaussians with the product of the exponential distributions in Equation 3, the likelihood is

$$p(x|S_l, \theta_l) = \prod_{s\in\Sigma_l} \Lambda_{ls} \lambda_{ls}^{c_{ls}(x)} e^{-\lambda_{ls} d_{ls}(x)} \ , \tag{3}$$

where

$$\Lambda_{ls} = \left(\frac{1}{\sigma_{ls}\sqrt{2\pi}}\right)^{c_{ls}(x)} e^{-\frac{1}{2\sigma_{ls}^2}(u_{ls}(x) - 2\mu_{ls} m_{ls}(x) + \mu_{ls}^2 c_{ls}(x))} \ . \tag{4}$$

Here, $u_{ls}(x) = \sum_{v \leftarrow s} v^2$, $m_{ls}(x) = \sum_{v\leftarrow s} v$ are the sufficient statistics where $v \leftarrow s$ indicates that v is the value of an event of the portion of x that has been mapped to state s. We add a normal-gamma distribution as the (independent) prior over each μ_{ls}-σ_{ls} pair of parameters to allow for closed-form Bayesian scores for structure learning via the greedy tree-growing algorithm.

3. Cohorts

Mortality modeling is often performed among heterogeneous population of patients to compare quality of care between different ICUs. To demonstrate the generality of our proposed modeling approach, we focus on two different cohorts, a cohort of pediatric patients in a tertiary PICU and a cohort of adult patients.

3.1 Cohort 1

We used data from the EHR archive of PICU at Children's Hospital Los Angeles (CHLA), which consists of 11684 patient episodes, collected over a period of 10 years. It includes demographics, outcomes, PIM2 and PRISM3 scores, and times and value for measurements of vital signs, interventions, drugs, and lab tests.

3.2 Cohort 2

We also conducted our experiments on the publicly available MIMIC-III database which integrates de-identified clinical data of patients admitted to the Beth Israel Deaconess Medical Center in Boston between 2001 and 2012. We removed patients younger than 18 years old to focus on adult patients in an ICU. The dataset includes vital signs, laboratory results, medications, charted observations, clinical notes.

4. Experiments

To show the effectiveness of our proposed temporal dependency modeling approach, we conducted experiments to predict in-hospital mortality based on the first 24 hours of clinical data for both the cohorts. The pre-processing step involves aggregating multiple related variables and normalization. Next, using a hold-out validation set, we select the final set of variables to be included in the PCIM models, based on the univariate performance of each variable. The final set of variables correspond to the set of event types or the label set L. Critical components of the models, such as the basis state functions, were also chosen using the validation set. Next, we learned two separate sets of PCIM models on the training set for the two classes of patients: patients who died in hospital and those who survived. Finally, we obtain a severity of illness score from the two sets of models using the log odds ratio and use this score as a mortality predictor. Details for each of the steps in our experimental process are explained in the following subsections.

4.1 Data Pre-Processing and Experimental Setup

For both datasets, some of the most commonly used clinical variables used in ICU scoring systems (PRISM3, SAPS-I, SAPS-II) were used to compose the primary sets of variables to perform our experiments. For the MIMIC-III dataset, we aggregated multiple related variables into one using a publicly available codebase[1]. The same codebase was also used to extract patient mortality outcomes, SAPS-I, SAPS-II score, and measurements for the first 24 hours. The data of Cohort 1 were similarly mapped.

For the cohort of pediatric patients, we normalized the values of age-dependent variables, such as heart rate, dividing by the median value among healthy children of the same age range and sex, according to published tables (Fleming et al., 2011). Then, we performed z-score normalization for all variables and removed data falling outside of ± 4 standard deviations from the mean of each variable. For adult patients there is relatively low variability on variables across different age groups and therefore we performed only z-score normalization.

The two class labels are "death" and "survival." We take the former to be the positive class. For both datasets, we randomly perform a (65-15-20) training-validation-test split. The validation set was used for hyper-parameter tuning and selection of the set of binary tests (both type and parameters). In order to make comparisons with PIM2 and PRISM3, we ran our final experiments for Cohort 1 on patient episodes with both PIM2 and PRISM3 scores reported in the dataset (PRISM3 scores of 0 were dropped). This resulted in a final dataset containing 4601 patient episodes (mortality rate 6.2%). Constructing only the test set with patient episodes having both scores reported would have introduced bias. For MIMIC-III, the final dataset included 34971 patient episodes corresponding to the first ICU and hospital stay of the patient (in hospital mortality rate 11.67%).

4.2 Choice of Binary Tests

The base binary tests available for internal nodes for the PCIM are listed in Table 1. The tests for checking whether the total number of measurements within some time interval is greater than a particular threshold (τ) is to allow the model to capture the burstiness in measurements of certain vital signs and lab tests. These tests allow modeling of self-excitation (burstiness) and self-suppression. We also introduce a test to check whether the number of measurements with value $\geq \tau'$ is greater than a particular threshold (τ) within a particular time interval. This allows the rate of subsequent events to depend on the values of previous events (e.g. measurements with extreme values of a particular variable causing more events of the same or a different variable in near future).

The full bank of binary tests are chosen by selecting the parameters from pre-determined sets of values. In particular, "past t hours" uses $t \in \{1/60, 1/12, 1/4, 1/2, 1, 24\}$; "value $\geq \tau'$" uses $\tau' \in \{-3.0, -2.5, -2, -1.5, 1.5, 2, 2.5, 3.0\} \times$std. dev.+mean; "number of measurements $\geq \tau$" uses $\tau \in \{0, 1, 5, 10, 20, 30, 50\}$; and "current time is within a particular interval of the day" uses range of one of "within δt minutes of the top of the hour," where $\delta t \in \{1, 2, 5, 15\}$ or within the first t hours of the day where $t \in \{1, 2, 3, \ldots, 24\}$.

4.3 Severity of Illness Score computation

Denote the set of models learned for the "died" class as M_{death} and the set for the "survived" class as M_{survive}. Both sets include PCIM models learned on the measurement timings and values of the selected variables. While each PCIM tree models the rate for one particular

1. https://github.com/MIT-LCP/mimic-code

number of measurements of variable X in past t hours is $\geq \tau$
number of measurements of variable X in past t hours with value $\geq \tau'$ is $\geq \tau$
current time t is within a particular interval within the day

Table 1: Binary Tests for learning PCIM. X denotes any measurement variable and τ and τ' are each one of a set of thresholds.

variable, the binary tests in the tree can look at all available variables (both vitals and labs) to capture the temporal dependencies with other variables. If the corresponding PCIM trees for a particular variable v in the two sets M_{death} and M_{survive} are denoted by m_{vd} and m_{vs} respectively then, the SOI score corresponding to variable v for patient with event sequence x is calculated by the log-odds: $\log \frac{p(m_{vd}|x)}{p(m_{vs}|x)}$. These individual SOI scores for each variable can be summed up to form a single SOI score or can be used as inputs along with other features to a final classifier model.

4.4 Full Severity Score

We use a feedforward neural network as the final classifier using the computed severity of illness scores (as above) and other features. For MIMIC-III, the static features include age, gender, minimum Glasgow Coma Scale, minimum ratio of partial pressure arterial oxygen and fraction of inspired oxygen, total urine output, co-morbidity, and reception of ventilation, all collected in the first 24 hours. For the CHLA dataset, we used PIM2 score as a feature.

For both datasets, we use a feedforward architecture with 3 hidden layers with 256 hidden units each. Number of hidden layers and number of hidden units were selected based on performance on validation data. A dropout (Srivastava et al., 2014) rate of 0.5 was used for each hidden layer. All the models were trained with the Adam optimization method (Kingma and Ba, 2014) using early stopping. We used the Keras Deep Learning Library (Chollet et al., 2015) for our implementations.

4.5 Baselines

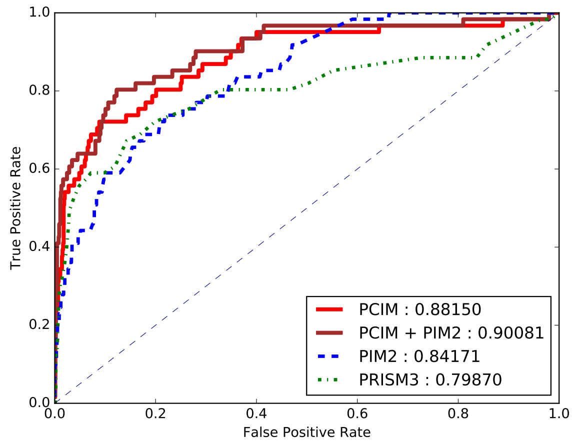
For evaluating the predictive performance of our approach, we compare with PIM2 and PRISM3 scores for cohort 1, and SAPS-I and SAPS-II scores for cohort 2.

5. Results and Discussion

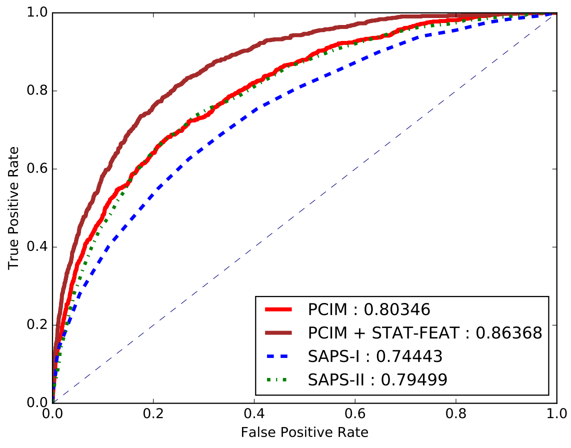
In this section, we present a comparative evaluation of our method with the standard ICU scoring systems. Statistical significance of the difference between the ROC curves presented was measured using MedCalc statistical software[2] (DeLong et al., 1988).

Figure 2(a) shows the comparison of ROC curves for PIM2, PRISM3, and SOI scores computed from PCIM models. SOI scores computed from the learned PCIM models outperform standard scoring system baselines, PIM2 and PRISM3 ($p(\text{PCIM} \sim \text{PRISM3}) < 0.024$ and $p(\text{PCIM} \sim \text{PIM2}) = 0.146$). The reason for relatively higher p-values is highly

2. https://www.medcalc.org

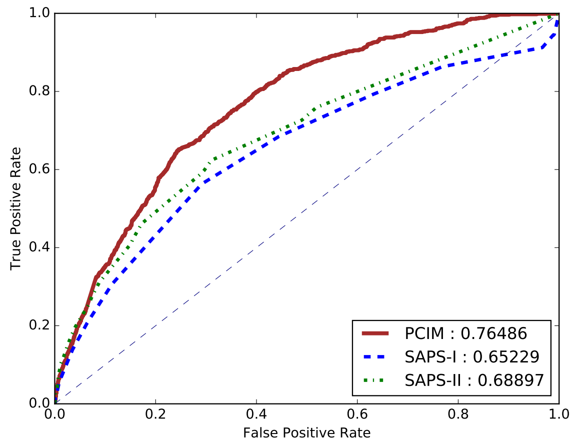


(a) Cohort 1 (CHLA)

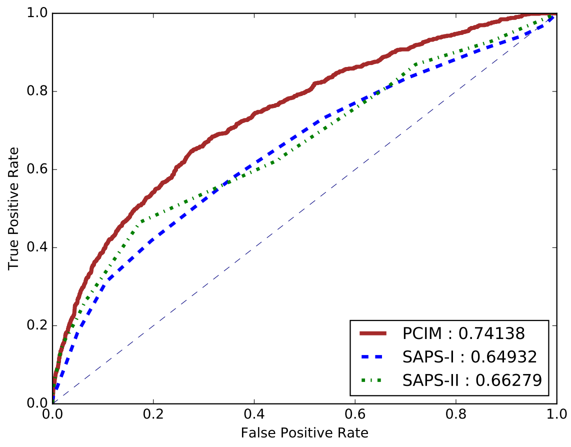


(b) Cohort 2 (MIMIC-III)

Figure 2: Comparison of all methods for in hospital mortality prediction



(a) Scores from lab variables only (MIMIC-III)



(b) Scores from vital signs only (MIMIC-III)

Figure 3: Comparison of all methods for the scoring of labs and vitals variables in MIMIC-III

likely to be caused by the small test set size of the Cohort 1. PCIM models do not directly incorporate information such as admission baseline features which capture how ill a child was at the time of admission. PIM2 is largely based on such features calculated within two hours of admission and outperforms PRISM3 in our experiments on the holdout testset. Best performance is achieved when PIM2 is combined with the SOI score obtained from the PCIM models by adding PIM2 as an extra feature to the feedforward net.

ROC curves for SAPS-I, SAPS-II, and SOI scores computed from PCIM models on the MIMIC-III dataset are presented in Figure 2(b). Similar to the CHLA dataset, pre-admission and in-ICU information, such as ventilation and comorbidity were not directly incorporated in the PCIM models. When added to our method (again, as features to the feedforward net), we outperform both SAPS-I and SAPS-II with statistical significance (AUROC difference has $p < 0.0001$).

For MIMIC-III, Figure 3 shows the ROC curves using only labs or only vital signs. Scores computed from the four vital variables significantly outperform (AUROC, $p < 0.0001$) the isolated vitals scores of both SAPS-I and SAPS-II, computed using certain ranges of values, Figure 3(b). The area under the ROC curve difference computed only using the lab variables is similarly statistically significant ($p(\text{PCIM} \sim \text{SAPS-I}) < 0.0001$ and $p(\text{PCIM} \sim \text{SAPS-II}) = 0.0001$), Figure 3(a).

6. Differences with existing Recurrent Neural Network based Methods

While existing RNN based approaches (Lipton et al., 2015; Aczon et al., 2017; Harutyunyan et al., 2017; Lipton et al., 2016; Che et al., 2016) can capture the temporal trend of the values of the variables, our proposed methodology also takes into account the rate of events by directly modeling the generation of a new event of a particular type with respect to previous events' values, timings, and types. One particular advantage to directly model the event generation process in continuous time domain is to avoid discrete window-based aggregation and any form of data imputation. Discrete time approaches usually assume the data are "missing at random," which does not always hold for irregularly sampled data. This can lead to reduced variability of more frequently measured signals. Imputing normal values for missing data loses the distinction between a truly normal and a missing measurement. Additionally, forward and back filling imputation strategies discard information about the timings of the measurements. Other advantages of our method are the model interpretability, added flexibility of model selection through the choice of basis state functions, and applicability in small to intermediate datasets, where neural network based methods might be an overkill and prone to overfitting. We would like to explore connections between recurrent neural networks and marked point process (Xiao et al., 2017; Du et al., 2016) in our future work to extract the advantages of both methods simultaneously.

7. Related Works

Standard ICU scoring systems developed for pediatric (e.g. PIM2, PRISM3) and adult (e.g. SAPS-I, SAPS-II) ICU units rely on scoring a patient based on a range of values of particular physiological variables recorded within the first 24 hours, in addition to information such as pre-ICU procedures, admission category and in-ICU ventilation data. However, they largely ignore the temporal nature of the data and are often susceptible to missing values. These scores are good for early detection of severe illness and are excellent benchmarking tools for determining ICU performance.

Recent work has focused on extracting more meaningful features from sources like clinical notes, which is ignored in traditional ICU scoring systems. Both Lehman et al. (2012) and Ghassemi et al. (2014) used non-parametric topic modeling techniques, hierarchical Dirichlet process and latent Dirichlet allocation (LDA) respectively, for learning the latent topic structure of nursing notes, which improves mortality prediction performance over SAPS-I and SAPS-II. Ghassemi et al. (2015) used multi-task Gaussian process (MTGP) models to model irregularly sampled clinical data and new clinical notes for acuity forecasting. Their approach shows that hyperparameters learned from MTGP models capturing the temporal change in topic membership of notes recorded for a particular patient can improve mortality prediction performance, if combined with SAPS-I and latent topic features. Jo et al. (2015) proposed a model combining LDA and Hidden Markov Models (HMMs) to capture the temporal dynamics of underlying patient states from the clinical notes and improve long term mortality prediction.

Lipton et al. (2015) used Long Short Term Memory Recurrent Neural Networks (LSTM-RNNs) in disease phenotyping. However, they employed a discrete (hourly) time window based approach and imputed values in missing windows using either forward or back-filling strategy. In a subsequent work (Lipton et al., 2016), better performance was achieved using binary indicator variables for the time windows with missing values, however it still relied upon a window based scheme. Aczon et al. (2017) developed a dynamic mortality risk prediction model using LSTM-RNNs in pediatric ICU data. They didn't resample the data at any fixed sampling rate, however depended upon zero or forward imputation for the values of variables, which had no recorded value at a particular time-point where at least one variable was recorded for a particular patient. Che et al. (2016) proposed a model based on Gated Recurrent Units (GRUs) with trainable decays to capture the temporal structure of the missing values. The AUROC achieved on the MIMIC-III dataset (19714 admission records) with 24 hours of data is 0.78821. The feature set however, consisted of 91 extra variables in addition to the 8 lab variables used separately in one of our experiments, achieving an AUROC of 0.76486. None of these works however, focused on capturing the intensities of events of varied types and their dependence on previous history across the timeline of a patient stay.

Marlin et al. (2012) focused on unsupervised learning from time series data collected from a pediatric ICU. While their approach shows non-uniformities among patients across different clusters extracted, their approach also employs a discrete time window and ignores measurement timings. Joshi and Szolovits (2012) modeled patient acuity using an unsupervised learning approach, radial domain folding, which in an organ specific manner, learns lower dimensional abstractions from routinely generated physiological data. Logistic regression models trained on the learned RDF layers outperform SAPS-II scoring system in mortality prediction. Weiss and Page (2013) used a forest-based point process model (an extension of the PCIM) to predict future onset of myocardial infractions. Based on the rates learned from event history, rates of future events are forecast. Their approach shows the strength of continuous time models in medical data, however doesn't address the temporal modeling of routinely measured physiological signals. Saria et al. (2010) focused on a non-parametric Bayesian method for data analysis in continuous time series including health care data, based on topic models.

8. Conclusion

Our contribution has been to demonstrate the efficacy of directly modeling the continuous-time temporal dependencies of discrete events recorded in an EHR. Our results indicate that a severity of illness score computed using our proposed modeling technique improves the performance of hospital mortality prediction.

One of the limitations of our approach is the use of a generative model for the task of classification, using the log-odds calculated from the representative models for the two class labels, learned separately. However, careful choice of the basis state functions from a validation set resulted in significant discrimination between learned models of the two classes and improved prediction performance. Discriminative training might better accentuate the temporal differences between the two classes. This is left as a future work.

Our method provided features to a feedforward neural network. We added other static measurements as features to the feedforward net. These could have also been added as possible values for binary tests in the PCIM model. This, particularly in conjunction with disciminative PCIM training, might work better as a classifier. We also focused only on nine variables for MIMIC-III because those are used in SAPS-II. The model might also be improved with an more expansive variable set. For the Cohort 1 data, we used more variables and saw a larger improvement. In one case, this was not statistically significant; we expect this is because of the relatively smaller sample size.

Acknowledgments

The authors gratefully thank the Virtual Pediatric Intensive Care Unit (vPICU) and the Laura P. and Leland K. Whittier Foundation for their help in this project and the collection of the CHLA data. This work was supported by DARPA (FA8750-14-2-0010).

References

Melissa Aczon, David Ledbetter, L Ho, Alec Gunny, Alysia Flynn, Jon Williams, and Randall Wetzel. Dynamic mortality risk predictions in pediatric critical care using recurrent neural networks. *arXiv preprint arXiv:1701.06675*, 2017.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *arXiv preprint arXiv:1606.01865*, 2016.

François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

Thomas L Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *AAAI*, pages 524–529, 1988.

Elizabeth R DeLong, David M DeLong, and Daniel L Clarke-Pearson. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, pages 837–845, 1988.

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564. ACM, 2016.

Susannah Fleming, Matthew Thompson, Richard Stevens, Carl Heneghan, Annette Plüddemann, Ian Maconochie, Lionel Tarassenko, and David Mant. Normal ranges of heart rate and respiratory rate in children from birth to 18 years of age: a systematic review of observational studies. *The Lancet*, 377(9770):1011–1018, 2011.

Nir Friedman, Michal Linial, Iftach Nachman, and Dana Pe'er. Using Bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, RECOMB '00, pages 127–135, New York, NY, USA, 2000. ACM. ISBN 1-58113-186-0. doi: 10.1145/332306.332355. URL `http://doi.acm.org/10.1145/332306.332355`.

Marzyeh Ghassemi, Tristan Naumann, Finale Doshi-Velez, Nicole Brimmer, Rohit Joshi, Anna Rumshisky, and Peter Szolovits. Unfolding physiological state: Mortality modelling in intensive care units. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 75–84. ACM, 2014.

Marzyeh Ghassemi, Marco AF Pimentel, Tristan Naumann, Thomas Brennan, David A Clifton, Peter Szolovits, and Mengling Feng. A multivariate timeseries modeling approach to severity of illness assessment and forecasting in ICU with sparse, heterogeneous clinical data. In *AAAI*, pages 446–453, 2015.

Asela Gunawardana, Christopher Meek, and Puyang Xu. A model for temporal dependencies in event streams. In *Advances in Neural Information Processing Systems*, pages 1962–1970, 2011.

Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *arXiv preprint arXiv:1703.07771*, 2017.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Yohan Jo, Natasha Loghmanpour, and Carolyn Penstein Rosé. Time series analysis of nursing notes for mortality prediction via a state transition topic model. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1171–1180, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3794-6. doi: 10.1145/2806416.2806541. URL `http://doi.acm.org/10.1145/2806416.2806541`.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3, 2016.

Rohit Joshi and Peter Szolovits. Prognostic physiology: modeling patient severity in intensive care units using radial domain folding. In *AMIA Annual Symposium Proceedings*, volume 2012, page 1276. American Medical Informatics Association, 2012.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jean-Roger Le Gall, Philippe Loirat, Annick Alperovitch, Paul Glaser, Claude Granthil, Daniel Mathieu, Philippe Mercier, Remi Thomas, and Daniel Villers. A simplified acute physiology score for ICU patients. *Critical care medicine*, 12(11):975–977, 1984.

Jean-Roger Le Gall, Stanley Lemeshow, and Fabienne Saulnier. A new simplified acute physiology score (SAPS II) based on a european/north american multicenter study. *Jama*, 270(24):2957–2963, 1993.

Li-Wei H Lehman, Mohammed Saeed, William J Long, Joon Lee, and Roger G Mark. Risk stratification of ICU patients using topic models inferred from unstructured progress notes. In *AMIA*. Citeseer, 2012.

Ted Petrie Leonard E. Baum. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966. ISSN 00034851. URL http://www.jstor.org/stable/2238772.

Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

Zachary C Lipton, David C Kale, and Randall Wetzel. Modeling missing data in clinical time series with rnns. *Machine Learning for Healthcare*, 2016.

Benjamin M. Marlin, David C. Kale, Robinder G. Khemani, and Randall C. Wetzel. Unsupervised pattern discovery in electronic health care data using probabilistic clustering models. In *Proceedings of the 2Nd ACM SIGHIT International Health Informatics Symposium*, IHI '12, pages 389–398, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0781-9. doi: 10.1145/2110363.2110408. URL http://doi.acm.org/10.1145/2110363.2110408.

Uri Nodelman, Christian R. Shelton, and Daphne Koller. Continuous time Bayesian networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, pages 378–387, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-897-4. URL http://dl.acm.org/citation.cfm?id=2073876.2073921.

Adam Oliner and Jon Stearley. What supercomputers say-an analysis of five system logs. In *IEEE/IFIP Conf. Dep. Sys. Net*, 2007.

Murray M Pollack, Kantilal M Patel, and Urs E Ruttimann. PRISM III: an updated pediatric risk of mortality score. *Critical care medicine*, 24(5):743–752, 1996.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989. ISSN 0018-9219. doi: 10.1109/5.18626.

Shyamsundar Rajaram, Thore Graepel, and Ralf Herbrich. Poisson-networks: A model for structured point processes. In *Proceedings of the 10th international workshop on artificial intelligence and statistics*, pages 277–284. Citeseer, 2005.

Suchi Saria, Daphne Koller, and Anna Penn. Learning individual and population level traits from clinical temporal data. In *Proc. Neural Information Processing Systems (NIPS), Predictive Models in Personalized Medicine workshop*. Citeseer, 2010.

Aleksandr Simma and Michael I. Jordan. Modeling events with cascades of poisson processes. *CoRR*, abs/1203.3516, 2012. URL `http://arxiv.org/abs/1203.3516`.

Aleksandr Simma, Moisés Goldszmidt, John MacCormick, Paul Barham, Richard Black, Rebecca Isaacs, and Richard Mortier. CT-NOR: representing and reasoning about events in continuous time. *CoRR*, abs/1206.3280, 2012. URL `http://arxiv.org/abs/1206.3280`.

Anthony Slater, Frank Shann, and Gale Pearson. PIM2: a revised version of the paediatric index of mortality. *Intensive Care Medicine*, 29(2):278–285, 2003. ISSN 1432-1238. doi: 10.1007/s00134-002-1601-2. URL `http://dx.doi.org/10.1007/s00134-002-1601-2`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2627435.2670313`.

Wilson Truccolo, Uri T Eden, Matthew R Fellows, John P Donoghue, and Emery N Brown. A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. *Journal of Neurophysiology*, 93(2):1074–1089, 2005.

Jeremy C Weiss and David Page. Forest-based point process for event prediction from electronic health records. In *Machine learning and knowledge discovery in databases*, pages 547–562. Springer, 2013.

Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M Chu. Modeling the intensity function of point process via recurrent neural networks. In *AAAI*, pages 1597–1603, 2017.

Appendix: Final Variables

Best motor response (GCS)	Best verbal response (GCS)
Glasgow coma scale total	Eye opening response(GCS)
Capillary refill rate (sec)	Diastolic blood pressure non-invasive (mmHg)
Systolic blood pressure invasive (mmHg)	Systolic blood pressure non-invasive (mmHg)
Pulse oximetry percentage	Heart rate (bpm)
Respiratory rate (bpm)	EtCO2 (mmHg)
Left pupillary response	Right pupillary response
Temparature (C)	

Table 2: Vital Signs (Cohort 1: CHLA)

ABG Base excess (mEq/L)	ABG PCO2 (mmHg)
ABG pH	Albumin level (g/dL)
BUN (mg/dL)	Bicarbonate serum (mEq/L)
Bilirubin total (mg/dL)	CBG PCO2 (mmHg)
CBG pH	Calcium ionized (mg/dL)
Calcium total (mg/dL)	Chloride (mEq/L)
Glucose (mg/dL)	P/F ratio
PT	PTT
Platelet count (K/uL)	Potassium serum (mEq/L)
Sodium serum (mEq/L)	VBG Base excess (mEq/L)
VBG PCO2 (mmHg)	VBG pH
White blood cell count (K/uL)	

Table 3: Labs (Cohort 1: CHLA)

Bicarbonate	BUN
Sodium	Potassium
WBC	Heart Rate
Respiratory Rate	Systolic Blood Pressure
Temperature(C)	

Table 4: Labs and Vital Signs (Cohort 2: MIMIC-III)

Appendix G

Event Detection in Continuous Video: An Inference in Point Process Approach

Appeared as

Zhen Qin and Christian R. Shelton. Event Detection in Continuous Video: An Inference in Point Process Approach. *IEEE Transaction on Image Processing*, 26(12), 5680–5691, 2017.

Event Detection in Continuous Video: An Inference in Point Process Approach

Zhen Qin and Christian R. Shelton

***Abstract*—We propose a novel approach towards event detection in real-world continuous video sequences. The method 1) is able to model arbitrary-order non-Markovian dependencies in videos to mitigate local visual ambiguities, 2) conducts simultaneous event segmentation and labeling, and 3) is time-window free. The idea is to represent a video as an event stream of both high-level semantic events and low-level video observations. In training, we learn a point process model called piecewise-constant conditional intensity model (PCIM) that is able to capture complex non-Markovian dependencies in the event streams. In testing, event detection can be modeled as the inference of high-level semantic events, given low-level image observations. We develop the first inference algorithm for PCIM and show it samples exactly from the posterior distribution. We then evaluate the video event detection task on real-world video sequences. Our model not only provides competitive results on the video event segmentation and labeling task, but also provides benefits including being interpretable and efficient.**

***Index Terms*—video event detection, event segmentation and labeling, video understanding, dependency modeling, video grammar, point process.**

I. Introduction

EVENT detection systems aim at identifying and localizing the classes of the events present in a video, such as a person sitting down, independently of the background. It is a key step towards real-world video understanding and has applications such as video indexing, video retrieval, and anomaly detection [33]. The large corpus of literature [34] usually model video event detection as a classification or labeling problem. Given coherent constituent parts from video segmentation in the temporal domain, a feature vector can be generated for each segment and serves as input for a discriminative classifier [18]. Another popular approach involves generating segmentation candidates via sliding windows and perform analysis at multiple temporal scales [20] [2].

However, video segmentation is an unsolved computer vision problem, and sliding windows approaches can be expensive. Also, visual ambiguity is unavoidable in real-world videos. By only looking at local visual features (either from a segmented clip or a time window), events of the same label might look quite different (when performed by different characters, or if the event has intrinsic intra-class variance), and events of different labels might look similar (for example, "punching" and "shaking hands" both consist of putting one's arm forward, see Fig. 1).

Z. Qin and C. Shelton are with the Department of Computer Science and Engineering, University Of California, Riverside, Riverside, CA, 92521 USA e-mail: {zqin001, cshelton}@cs.ucr.edu.

This work was supported by the Defense Advanced Research Projects Agency (FA8750-14-2-0010) and the National Science Foundation (IIS 1510741).

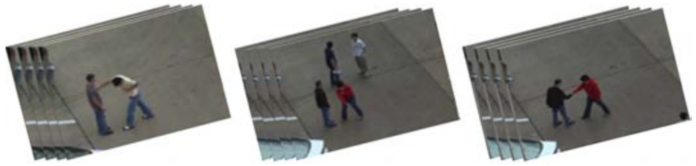


Fig. 1: (Left & Middle) Punching, (Right) Shaking Hand. Based only on visual features, the same action can look different, while different actions can possess similar appearances.

Contextual information could help to disambiguate, and we focus on modeling temporal context in this work. For example, if followed by the "person running" or "person falling down" events, we should be more certain that the event before is "punching", instead of "shaking hand".

We propose a new approach to explore temporal dependencies among events and visual observations in video: In training, given both observed low-level events (local visual features) and annotated high-level semantic events, we can build a point process model to learn complex dependencies in video event streams. In testing, the detection of high-level events can be modeled as an inference problem, given the observed low-level events. See Fig. 2 for an illustration of an event stream representation of video. This modeling approach allows us to leverage the machine learning and statistics literature on dependency modeling in point process.

We use a state-of-art point process model, called a piecewise-constant conditional intensity model (PCIM) [17]. PCIM captures the dependencies among the types of events through a set of piecewise-constant conditional intensity functions. A PCIM is represented as a set of decision trees (see Fig. 3 for an example), which provide model interpretability and allow for efficient model selection. In training, by applying PCIM learning on annotated videos, PCIM is able to learn the complex dependencies in the (annotated) video event streams, with the extra benefit of providing a meaningful video grammar.

In testing, the video event detection task we are interested in requires an inference algorithm for PCIM. An inference algorithm allows localizing and labeling high-level semantic events given only low-level visual observations in unannotated testing videos. An exact inference algorithm is able to take advantage of the rich dependencies learned in training, thus mitigating local visual ambiguities in video event detection. Also, inference in point process provides both time and label of inferred events, allowing automatic simultaneous event segmentation and labeling, without the need of sliding windows. However, no inference algorithm has been proposed for

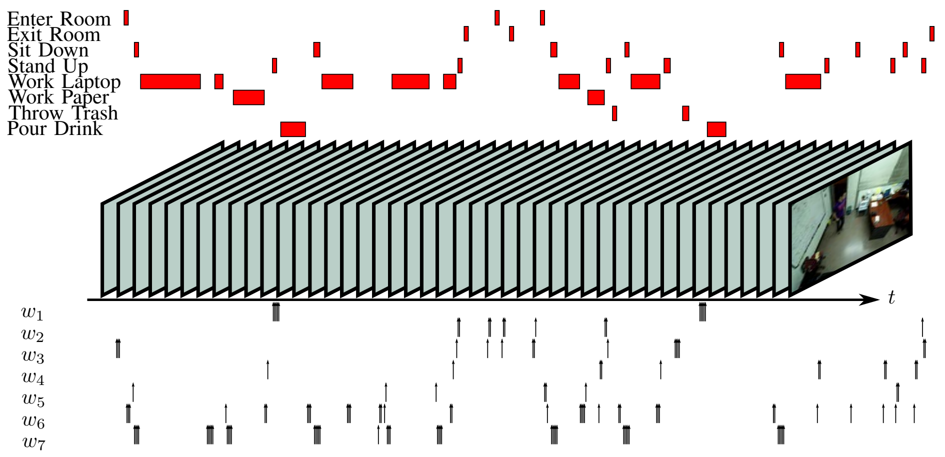


Fig. 2: A sample event stream representation of video. The events above the video clip are the semantic high-level events. ws are the low-level visual observations. Note for high-level events we use bars for a clearer illustration, in practice each of them is represented as two points (starting time and ending time with event labels). See text for more details.

PCIM that can condition on evidence (visual observations). Correctly filling in incomplete event streams from a PCIM is challenging, due to the complex non-Markovian dependencies between past and future evidence.

In this paper, we propose the first general inference algorithm for PCIMs, based on *thinning* for inhomogeneous Poisson process [26]. This inference algorithm can be used in the video event detection task, as well as any other tasks using PCIMs. Our formulation is an auxiliary Gibbs sampler that alternates between sampling a finite set of virtual event times given the current trajectory, and then sampling a new trajectory given the set of evidences and event times (virtual and actual). Our method is convergent, does not involve approximations like fixed time-discretization, and the samples generated can answer any type of query. We propose an efficient state-vector representation to maintain only the necessary information for diverging trajectories, reducing the exponentially increasing sampling complexity to linear in most cases. We show empirically our inference algorithm converges to the true distribution and permits effective query answering for PCIM models with both Markovian and complex non-Markovian dynamics.

We apply the PCIM inference algorithm to the video event detection task and show competitive results over state-of-arts on real-world long continuous videos. Our modeling approach is able to learn complex temporal dependencies in video, and exploit these dependencies to mitigate local visual ambiguities in event detection. The major contributions of our work are:

- A novel approach for video event segmentation and labeling via inference in point process, which does not rely on video pre-segmentation or sliding windows;
- The use of a PCIM to learn complex dependencies in video and provide meaningful video grammars; and
- The first exact inference algorithm for PCIMs that can be used for event detection and other tasks. [35] described a preliminary piece of this inference algorithm.

A. Related Work

We first describe related work in machine learning that models temporal dependencies. A dynamic Bayesian network (DBN) [10] models temporal dependencies between variables in discrete time. Continuous-time models have drawn attention recently in applications ranging from social networks [11][13] to genetics [8] to biochemical networks [15]. Continuous Time Bayesian Networks (CTBN) [29] are homogeneous *Markovian* models of the joint trajectories of discrete finite variables, analogous to DBNs. Non-Markovian continuous models allow the rate of an event to be a function of the process's history. Poisson Networks [36] constrain this function to depend only on the counts of the number of events during a finite time window. Hawkes processes [19] define the rate to be the sum of a kernel applied to each historic event, requiring the modeler to choose the form for the kernel.

A PCIM defines the intensity function as a decision tree, with internal nodes' tests mapping time and history to leaves. Each leaf is associated with a constant rate. A PCIM is able to model non-Markovian temporal dependencies, and is an order of magnitude faster to learn than Poisson networks. Successful applications include modeling supercomputer event logs and forecasting future interests of web search users [17]. While PCIMs have drawn attention recently [31] [44] and have potential usage in a wide variety applications, there is no general inference algorithm.

Inference algorithms developed for continuous systems are mainly for Markovian models (or specifically designed for a particular application). For CTBNs, there are variational approaches such as expectation propagation [12] and mean field [8], which do not converge to the true value as computation time increases. Sampling based approaches include importance sampling [14] and Gibbs sampling [37] [38] that converge to the true value. The latter is the current state-of-the-art method designed for general Markov Jump Processes (MJPs) and its extensions (including CTBNs). It uses the idea of uniformization [16] for Markov models, similar to thinning [26] for inhomogeneous Poisson processes. We note that our inference method for PCIM generalizes theirs to non-Markovian models.

Modeling temporal dependencies in general event streams has wide applications. For example, users' behaviors in online shopping [45] and web searches [27], as well as electronic health records [44] can all be viewed as a stream of events over time. Event streams in video is a specific case of general event streams.

To identify and recognize events or actions in video, computer vision researchers mainly focused on the classification or labeling problem given pre-segmented video clips [34]. Feature representations used in the literature are generated using local features such as Space Time Interest Points (STIP) [24], Dense Trajectories [41], Fisher Vectors [32], and more recently, deep learned representations [18] [21] [42] [47], among many others. Some work explore contextual information and temporal structure within complex video events [1] [40] [48] [23], largely motivated by the popular TRECVID MED challenge [30]. Our model learns dependencies both within and among high-level events, and can also be applied to the complex event detection task.

However, real-world videos are continuous, and the task of video segmentation is an unsolved problem [9]. Also, most existing methods entail shot boundary detection, i.e., the segmentation of a video into continuously imaged (usually in terms of motion) temporal segments, which rarely maps to individual high-level events [5]. Recently, some work tries to address the problem of simultaneous video segmentation and labeling [20] [7] [6]. These methods mostly use the time-consuming sliding window approaches, which process each segment independently at multiple time scales. Post-processing such as duration priors and non-max suppression is required [6]. It is very difficult for these approaches to handle local visual ambiguities in real-world videos, since each event/video segment is processed independently from the others. High-level contexts at video level, such as temporal dependencies, have rarely been explored.

Some work explores Markovian dependencies among events [22], which is limited for real-world videos with complex dependencies. [50] models dependencies among events in a video based on Conditional Random Fields (CRF). However, it can only explore dependencies up to some fixed order (chosen manually), and the computation becomes infeasible when the order of dependency specified increases. Also, it assumes a long video has been segmented before doing dependency modeling. Recently, recurrent Neural Network (RNN) based methods are drawing more attention [46]. They focus more on sequence dependency modeling, instead of temporal modeling. Temporal modeling using PCIM has the advantage of being able to easily model cases such as "someone enters the office around 8am everyday", regardless of (potentially unbounded number of) events happened in between. Explicitly modeling time can also be beneficial in real-world videos with timestamps (such as videos from surveillance cameras). It is also easier for humans to interpret and generate rules from the decision tree representation of PCIM. Furthermore, RNN based approaches tend to be training data hungry. Our idea of using event stream models explicitly address temporal dependencies in the continuous-time domain and is the first to do so, to the best of our knowledge.

II. Background on PCIM

We first briefly review the background on Piecewise-constant conditional intensity model (PCIM).

Assume events are drawn from a finite label set L. An event then can be represented by a time-stamp t and a label l. An event sequence $x = \{(t_i, l_i)\}_{i=1}^n$, where $0 < t_1 < \ldots < t_n$. We use $h_i = \{(t_j, l_j) \mid (t_j, l_j) \in x, t_j < t_i)\}$ for the history of event i, when it is clear from context which x is meant. We define the ending time $t(y)$ of an event sequence y as the time of the last event in y, so that $t(h_i) = t_{i-1}$. A conditional intensity model (CIM) is a set of non-negative conditional intensity functions indexed by label $\{\lambda_l(t|x;\theta)\}_{l=1}^{|L|}$. The data likelihood is

$$p(x|\theta) = \prod_{l\in L}\prod_{i=1}^{n} \lambda_l(t_i|h_i;\theta)^{\mathbf{1}_l(l_i)} e^{-\Lambda_l(t_i|h_i;\theta)} \tag{1}$$

where $\Lambda_l(t|h;\theta) = \int_{t(h)}^{t} \lambda_l(\tau|h;\theta)d\tau$. The indicator function $\mathbf{1}_l(l^{'})$ is one if $l^{'} = l$ and zero otherwise. $\lambda_l(t|h;\theta)$ is the expected rate of event l at time t given history h and model parameters θ. Conditioning on the entire history causes the process to be non-Markovian. The modeling assumptions for a CIM are quite weak, as any distribution for x in which the timestamps are continuous random variables can be written in this form. Despite the weak assumptions, the per-label conditional factorization allows the modeling of label-specific dependence on past events.

A PCIM is a particular class of CIM that restricts $\lambda(h)$ to be piecewise constant (as a function of time) for any history, so the integral for Λ breaks down into a finite number of components and forward sampling becomes feasible. A PCIM represents the conditional intensity functions as decision trees. Each internal node in a tree is a binary test of the history, and each leaf contains an intensity. If the tests are piecewise-constant functions of time for any event history, the resulting function $\lambda(t|h)$ is piecewise-constant. Examples of admissible tests include

- Was the most recent event of label l?
- Is the time of the day between 6am and 9am?
- Did an event with label l happen at least n times between 5 seconds ago and 2 seconds ago?
- Were the last two events of the same label?

Note some tests are non-Markovian in that they require knowledge of more than just which event was most recent. See Fig. 3 for an example of a PCIM model.

The decision tree for label l maps the time and history to a leaf $s \in \Sigma_l$, where Σ_l is the set of leaves for l. The resulting data likelihood can be simplified:

$$p(x|S,\theta) = \prod_{l\in L}\prod_{s\in\Sigma_l} \lambda_{ls}^{c_{ls}(x)} e^{-\lambda_{ls} d_{ls}(x)}. \tag{2}$$

S is the PCIM structure represented by the decision trees; the model parameters θ are rates at the leaves. $c_{ls}(x)$ is the number of times label l occurs in x and is mapped to leaf s. $d_{ls}(x)$ is the total duration when the event trajectory for l is mapped to s. Together, c and d are the sufficient statistics for calculating the likelihood.

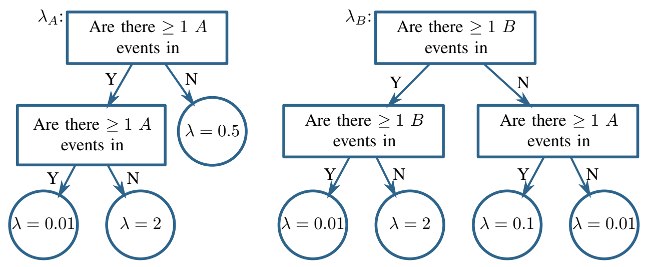

λ_A:
Are there ≥ 1 A events in
Y
N
Are there ≥ 1 A events in
$\lambda = 0.5$
Y
N
$\lambda = 0.01$
$\lambda = 2$
λ_B:
Are there ≥ 1 B events in
Y
N
Are there ≥ 1 B events in
Are there ≥ 1 A events in
Y
N
Y
N
$\lambda = 0.01$
$\lambda = 2$
$\lambda = 0.1$
$\lambda = 0.01$

Fig. 3: Decision tree representing S and θ for events of labels A and B. Note the dependency among event labels (the rate of B depends on A). [17]

[17] showed that given the structure S, by using a product of Gamma distributions as a conjugate prior for θ, the marginal likelihood of the data can be given in closed form, and thus parameter estimation can be done in closed form. The prior density is given by

$$p(\lambda_{ls}|\alpha_{ls}, \beta_{ls}) = \frac{\beta_{ls}^{\alpha_{ls}}}{\Gamma(\alpha_{ls})} \lambda_{ls}^{\alpha_{ls}-1} e^{-\beta_{ls}\lambda_{ls}}, \tag{3}$$

and the posterior density is given by

$$p(\lambda_{ls}|\alpha_{ls}, \beta_{ls}, x) = p(\lambda_{ls}|\alpha_{ls} + c_{ls}(x), \beta_{ls} + d_{ls}(x)). \tag{4}$$

Assuming the prior over the model parameters θ is a product of such priors, the marginal likelihood of data is

$$p(x|S) = \prod_{l \in L} \prod_{s \in \Sigma_l} \gamma_{ls}(x), \tag{5}$$

with

$$\gamma_{ls}(x) = \frac{\beta_{ls}^{\alpha_{ls}}}{\Gamma(\alpha_{ls})} \frac{\Gamma(\alpha_{ls} + c_{ls}(x))}{(\beta_{ls} + d_{ls}(x))^{\alpha_{ls}+c_{ls}(x)}}. \tag{6}$$

Then the authors choose to use a simple point estimate $E[\lambda_{ls}|x]$ for the rate, which is $\frac{\alpha_{ls}+c_{ls}(x)}{\beta_{ls}+d_{ls}(x)}$.

Furthermore, imposing a structural prior allows a closed form Bayesian score to be used for greedy tree learning. The local structure S_l can be chosen independently for each l by using a factored structural prior

$$p(S) \propto \prod_{l \in L} \prod_{s \in \Sigma_l} \kappa_{ls} \tag{7}$$

and the prior and the marginal likelihood that also factor over l. Given the current structure S_l (initialized as a single root), a new structure $S_l^{'}$ is considered by choosing a leaf s and expand it with a test to get a set of new leaves $\{s_1, \cdots, s_m\}$. The gain in the posterior of the structure is

$$\frac{p(S_l^{'}|x)}{p(S_l|x)} = \frac{\kappa_{ls_1}\gamma_{ls_1}(x) \cdots \kappa_{ls_m}\gamma_{ls_m}(x)}{\kappa_{ls}\gamma_{ls}(x)}. \tag{8}$$

The new structure with the largest gain is chosen if the gain is larger than 1.

III. Event Detection in Video as Inference in Point Process

We apply PCIM to event localization and labeling in video. We first show how to represent videos as event streams. A PCIM can be learned from training videos, represented by event streams, to encode nonlinear temporal dependencies between high-level events and low-level observations. In testing, an inference algorithm can be used to infer high-level events given low-level visual observations. This framework is among the first to encode temporal context for the event detection in long continuous video task.

A. Representation

Assume there are M high-level events in a video dataset, each of which is an event with high-level semantics, such as “a person sitting down” and “a person working on a laptop”. We generate $2M$ event labels to be used in PCIM: $\{s_1, \ldots, s_M\}$ and $\{e_1, \ldots, e_M\}$. s_i indicates the starting of a high-level event type i and e_i indicates the ending of a high-level event type i. These are the event types that are labeled in training and to be inferred in testing.

Given a video, we divide it into segments of fixed length, and a feature vector is generated for each of the segments. This step is flexible, any existing video descriptors might be applied here. Then we learn a dictionary (e.g. using K-means clustering), so that each segment can be assigned to one visual word in $\{w_1, \ldots, w_K\}$ and a time (we use the middle time of each segment in the video). Together with the starting and ending of high-level events, we have an event stream representation of a video. See Fig. 4 for an example.

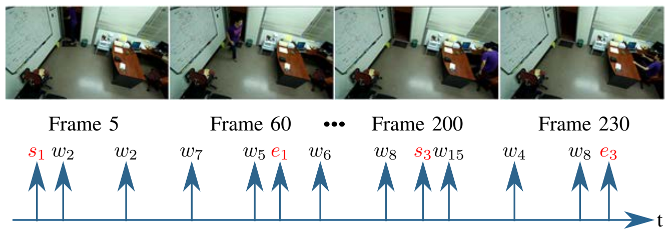

Frame 5
Frame 60
•••
Frame 200
Frame 230
s_1 w_2 w_2 w_7 w_5 e_1 w_6 w_8 s_3 w_{15} w_4 w_8 e_3
t

Fig. 4: An illustration of an event stream representation for video. Low-level observations are represented as regularly spaced events from a dictionary (the ws). s_1 and e_1 indicate the starting and ending of “entering room”. s_3 and e_3 indicate the starting and ending of “sitting down”.

There are several benefits for this representation: First, by using fixed-length segment, we do not assume semantic video pre-segmentation. Second, the usage of starting and ending of high-level events enables automatic temporal localization and labeling. Note that even though the low-level visual words are regularly sampled, each word is sparse across the timeline, which is suitable for a continuous-time model.

B. Training

Given an event stream representation of video, training can be done by directly feeding the training data into the PCIM learning algorithm. The resulted PCIM encodes temporal dependencies between both high-level events and low-level visual observations. There are three types of dependencies a PCIM can learn:

Dependency between high-level events. Global dependencies between high-level events can be modeled, which helps to mitigate visual ambiguities by utilizing temporal context. For example, after a person working on laptop, the probability of a person standing up should be higher than the person sitting down. PCIM is also able to learn the dependency between s and e for each high-level events, which encodes the temporal range distribution of each event type.

Dependency between high-level and low-level events. This type of dependency can be treated as local dependency. Certain low-level observations indicate the appearance of high-level events, or as a generative model, the high-level events "cause" low-level features. In testing, low-level visual words are observed, and are responsible for proposing high-level events.

Dependency between low-level events. This kind of dependency provides interesting information about how low-level observations can be correlated from a video grammar perspective. But for the event detection problem, it is not very useful as in testing all low-level observations are observed.

C. Testing as Inference

Given a PCIM learned from training data, we can model the problem of event detection in video as the inference of high-level events, given low-level observations. In other words, all the ws are observed, while the ss and es are completely unobserved. We can then apply our new inference algorithm, ThinnedGibbs, detailed in Sec. IV, to infer the starting and ending times of the high-level events. See Fig. 5 and Fig. 6 for an illustration.

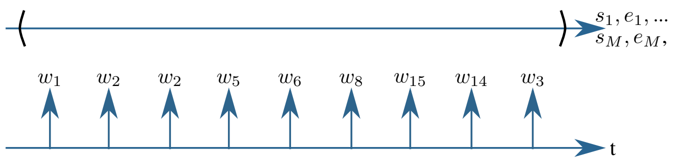


Fig. 5: In testing, the low-level events are fully observed, while the high-level events are not observed (in parentheses).

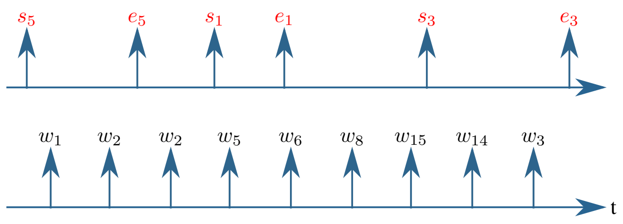


Fig. 6: After running inference, each sample would fill in the unobserved intervals for high-level events, which indicate their starting and ending times.

IV. Auxiliary Gibbs Sampling for PCIM

In this section we introduce our new inference algorithm for PCIM, called ThinnedGibbs, based on the idea of *thinning* for inhomogeneous Poisson processes. We handle incomplete data in which there are intervals of time during which events for particular label(s) are not observed.

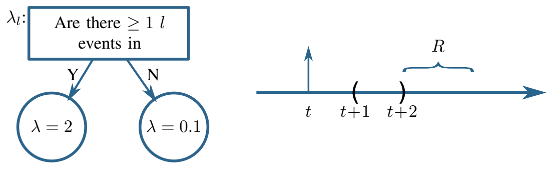


Fig. 7: A simple PCIM with a partially observed trajectory. The vertical solid arrow indicates an evidence event. Areas between parentheses are unobserved. History alone indicates there should be events filled in, while the future (no events in R) provides contradictory evidence.

A. Why Inference in PCIM is Difficult

Filling in partially observed trajectories for PCIM is hard due to the complex dependencies between unobserved events and both past and future events. See Fig. 7 for an example. While the history (the event at t) says it is likely that there should be events in the unobserved area (with an expected rate of 2), future evidence (no events in R) is contradictory: If there were indeed events in the unobserved area, those events should stimulate events happening in R.

Such a phenomenon might suggest existing algorithms such as the forward-filtering-backward-sampling (FFBS) algorithm for discrete-time Markov chains. However, there are two subtleties here: First, we are dealing with non-Markovian models. Second, we are dealing with continuous-time systems, so the number of time steps over which to propagate is infinite.

B. Thinning

Thinning [26] can be used to turn a continuous-time process into a discrete-time one, without using a fixed time-slice granularity. We select a rate λ^* greater than any in the inhomogeneous Poisson process and sample from a *homogeneous* process with this rate. To get a sample from the original inhomogeneous process, an event at time t is thinned (dropped) with probability $1 - \frac{\lambda(t)}{\lambda^*}$.

This process can also be reversed. Given the set of thinned event times (from the inhomogeneous process), the extra events can be added by sampling from a Poisson process with rate $\lambda^* - \lambda(t)$. The cycle can then repeat by thinning the new total set of times. At each cycle, the times (after thinning) are drawn from the original inhomogeneous process. We will use this type of cycle in our sampler.

The difficulty is that a PCIM is not an inhomogeneous Poisson process. The intensity depends on the entire history of events, not just the current time. For thinning, this means that we cannot independently sample whether each event is to be thinned. Furthermore, we wish to sample from the posterior, conditioned on evidence. All evidence (both past and future) affect the probability of a specific thinning configuration.

C. Overview of Our Inference Method

To overcome both of these problems, we extend thinning to an auxiliary Gibbs sampler in the same way that [37], [38] extended Markovian-model uniformization [16] (a specific example of thinning in a Markov process) to a Gibbs sampler.

To do this we introduce auxiliary variables representing the events that were dropped. We call these events *virtual* events.

As a standard Gibbs sampler, our method cycles through each variable in turn. In our case, a variable corresponds to an event label. For event label l, let x_l be the sampled event sequence for this label. Let Y_l be all evidence (for l and other labels) and all (currently fixed) samples for other labels. Our goal is to sample from $p(x_l \mid Y_l)$.

Let v_l be the virtual events (the auxiliary variable) associated with l and $z_l = x_l \cup v_l$ (all event times virtual and non-virtual). Our method first samples from $p(v_l \mid x_l, Y_l)$ and then samples from $p(x_l \mid z_l, Y_l)$. The first step adds virtual events given the non-virtual events are "correct." The second step treats all events as potential events and drops or keeps events. The dropped events are removed completely. The kept events, x_l, remain as the new sampled trajectory for label l.

The proof of correctness follows analogously to that of [38] for Markovian systems. But, the details for sampling from $p(v_l \mid x_l, Y_l)$ and $p(x_l \mid z_l, Y_l)$ differ. We describe them next.

D. Sampling Auxiliary Virtual Events with Adaptive Rates

Sampling from $p(v_l \mid x_l, Y_l)$ amounts to adding just the virtual (dropped) events. As the full trajectory (x_l for all l) is known, the rate at any time step for a virtual event is independent of any other virtual events. Therefore, the process is an inhomogeneous Poisson process for which the rate at t is equal to $\lambda^* - \lambda_l(t|h)$ where h is fully determined by x_l and Y_l. Recall that $\lambda_l(t|h)$ is piecewise-constant in time, so sampling from such an inhomogeneous Poisson process is simple.

The auxiliary rate, λ^*, must be strictly greater than the maximum rate possible for irreducibility. We use an auxiliary rate of $\lambda^* = 2\max(\lambda(t|h))$ to sample virtual events in the unobserved intervals. This choice trades off well between mixing time and computational complexity in the experiments.

A naïve way to pick λ^* is to find λ_{max}: the maximum rate in the leaves of PCIM, and use $2\lambda_{max}$. However, there could be unobserved time intervals with a possible maximum rate much smaller than λ_{max}. Using λ_{max} in those regions would generate too many virtual events, most of which will be dropped in the next step leading to computational inefficiency. We therefore use an adaptive strategy.

Our adaptive $\lambda^*(t|h)$ cannot depend on x_l (this would break the simplicity of sampling mentioned above). Therefore, we determine $\lambda^*(t|h)$ by passing (t, h) down the PCIM tree for λ_l. At each internal node, if the branch does not depend on x_l, we can directly take one branch. Otherwise, the test is related to the sampled events, and we take the maximum rate of taking both branches. This method results in $\lambda^*(t|h)$ as a piecewise-constant function of time (for the same reasons that $\lambda_l(t|h)$ is piecewise-constant).

Consider Fig. 8 as an example. When sampling event $l = A$ on the interval $[1, 5)$, we would not take the left branch at the root (no matter what events for A have been sampled), but must maximize over the other two leaves (as different x_l values would result in different leaves). This results in a $\lambda^* = 4$ over this interval, which is smaller than 6.

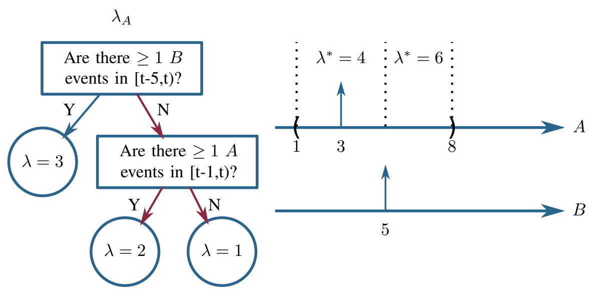


Fig. 8: Adaptive auxiliary rate example. When sampling A, the branch to take at the root does not depend on unobserved events for A. If the test is related to the sampled event, we take the maximum rate from both branches. The red arrows indicate the branches to take between time $[1, 5]$, and $\lambda^* = 2 \times 2$ in that interval, instead of 6.

E. The Naïve FFBS Algorithm

Once these virtual events are added back in, we take z_l (the union of virtual and "real" sampled events) as a sample from the Poisson process with rate λ^*, ignoring which were originally virtual and which were originally "real." We thin this set to get a sample from the conditional marginal over l.

The restriction to consider events only at times in z_l transforms the continuous-time problem into a discrete one. Given z_l with m possible event times $(z_{l,1}, z_{l,2}, \ldots, z_{l,m})$, let $b = \{b_i\}_{i=1}^m$ be a set of binary variables, one per event, where $b_i = 1$ if event i is included in x_l (otherwise $b_i = 0$ and the event is not included in x_l). Thus sampling b is equivalent to sampling x_l (z_l is known) as it specifies which events in z_l are in x_l. Let $Y_l^{i:j}$ be the portion of Y between times $z_{l,i}$ and $z_{l,j}$, and $b^{i:j} = \{b_k | i \leq k \leq j\}$ We wish to sample b (and thereby x_l) from

$$p(b \mid Y) \propto \left(\prod_i p(Y_l^{i-1:i}, b_i \mid b^{1:i-1}, Y_l^{1:i-1}) \right) p(Y_l^{m:\infty} \mid b) \tag{9}$$

where the final $Y_l^{m:\infty}$ signifies all of the evidence after the last virtual event time $z_{l,m}$ and can be handled similarly to the other terms.

The most straight-forward method for such sampling considers each possible assignment to b (of which there are 2^m). For each interval, we multiply terms from Eq. 9 of the form $p(Y_l^{i-1:i}, b_i \mid b^{1:i-1}, Y_l^{1:i-1}) =$

$$p(Y_l^{i-1:i} \mid b^{1:i-1}, Y_l^{1:i-1}) p(b_i \mid b^{1:i-1}, Y_l^{1:i}) \tag{10}$$

where the first term is the likelihood of the trajectory interval from $z_{l,i-1}$ to $z_{l,i}$ and the second term is the probability of the event being thinned, given the past history. The first can be computed by tallying the sufficient statistics (counts and durations) and applying Eq. 2. Note that these sufficient statistics take into account $b^{1:i-1}$ which specifies events for l during the unobserved region(s), and the likelihood must also be calculated for labels $l' \neq l$ for which $\lambda_{l'}(t|h)$ depends on events from l. The second term is equal to $\frac{\lambda_l(t|h)}{\lambda^*(t)}$ if $b_i = 1$ (and $1 - \frac{\lambda_l(t|h)}{\lambda^*(t)}$ if $b_i = 0$). The numerator's dependence on he full history similarly dictates a dependence on $b^{1:i-1}$.

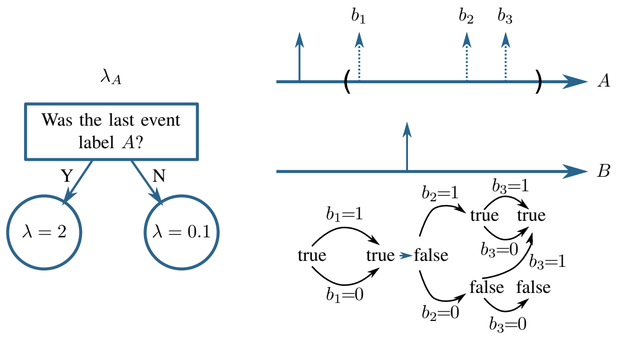


Fig. 9: Dotted events are the virtual events that we sample as binary variables (b_i is 1 if event i is kept). The state diagram below the trajectory indicates the state of the test as we diverge (keep or drop a virtual event). Though there are 2^3 possible configurations, state merges can reduce the exponentially increasing complexity to linear in this case.

This might be formulated as a naïve FFBS algorithm: To generate one sample, we propagate possible trajectories forward in time, multiplying in Eq. 10 at each inter-event interval to account for the evidence. Every time we see a virtual event, each possible trajectory diverges into two (depending on whether the virtual event is to be thinned or not). By the end, we have all 2^m possible trajectories, each with its probability (Eq. 9). We sample one trajectory as the output, in proportion of the calculated likelihoods. As we explicitly keep all possible trajectories, the sampled trajectory immediately tells us which virtual events are kept, so no actual backward pass is needed.

F. An Efficient State-Vector Representation

The naïve FFBS algorithm is not practical, as the number of possible trajectories grows exponentially with the number of auxiliary virtual events, m. We propose a more efficient state-vector representation to only keep the necessary information for each possible trajectory. The idea takes advantage of the structure of the PCIM and leads to state merges, similar to what happens in FFBS for hidden Markov models (HMMs).

The terms in Eq. 10 depend on $b^{1:i-1}$ only through the tests in the internal nodes of the PCIM trees. Therefore, we do not have to keep track of all of $b^{1:i-1}$ to calculate these likelihoods, but only the current state of such tests that depend on events with label l. For example, a test that asks "Is the last event of label l?" only needs to maintain a bit as the indicator. The test "Are there more than 3 events of label q in the last 5 seconds?" for $q \neq l$ has no state, as $b^{1:i-1}$ does not affect its choice. By contrast, a test such as "Is the last event of label q?" does depend on b, even if $q \neq l$.

As we propagate forward, we merge $b^{1:i}$ sequences that result in the same set of states for all internal tests inside the PCIM. See Fig. 9 as a simple example. Though there are 8 possible trajectories, they merge to only 2 states that we can sample from. Similar to the FFBS for an HMM, we need to maintain the transition probabilities in the forward pass and use them in a backward sampling pass to recover the full trajectory, but such information is also linear.

Note that this conversion to a Markov system for sampling is *not* possible in the original continuous-time system. Thinning allowed this by randomly selecting a few discrete time points and thereby restricting the possible state space to be finite.

The state space depends on the actual tests in the PCIM model. See Tbl. I for the tests we currently support and their state representations. The LastStateTest and StateTest are used to support discrete finite variable systems, such as a CTBN. We can see that for tests that only depend on the *current* time (i.e. TimeTest), the diverging history does not affect them, so no state is needed. For Markovian tests (LastEventTest and LastStateTest), we only need a Boolean variable. For the non-Markovian test (EventCountTest), the number of possible states does grow exponentially with the number of virtual events maintained in the queue. This is the best we can do and still be exact. It is much better than growing with the number of all virtual events. However, note that commonly $lag2 = 0$ and n is small. In this case, the state space size at any point is bounded as $\binom{m'}{n}$, where m' is the maximum number of sampled events in any time interval of duration $lag1$ (which is upper bounded by m). If n is 1, this is linear in the number of samples generated in during $lag1$ time units.

As noted above, if the test is not related to the sampled event (for example, we are sampling event $l = A$ and the test is "are there ≥ 3 B events in the last 5 seconds?"), the state of the test is set to null. This is because the evidence and sampled values for B (which is not the current variable for Gibbs sampling) can answer this test without reference to samples for l.

See Alg. 1 for the algorithm description for resampling event l. The complete algorithm iterates this procedure for each event label to get a new sample. The helper function UpdateState(s,b,t) returns the new state given the old state (s), the new time (t), and whether an event occurs at t (b). SampProbMap(M) takes a mapping from objects to positive values (M) and randomly returns one of the objects with probability proportional to the associate value. AddtoProbMap(M,o,p) checks to see if o is in M. If so, it adds p to the associated probability. Otherwise, it adds the mapping $o \rightarrow p$ to M.

G. Extended Example

Fig. 10 shows an example of resampling the events for label A on the unobserved interval $[0.8, 3.5)$. On the far left is the PCIM rate tree for event A. Box (a) shows the sample from previous iteration (single event at 2.3). Dashed lines and λ show the piecewise-constant intensity function given the sample. Box (b) shows the sampling of virtual events. For this case $\lambda^* = 3$ for all time. $\lambda^* - \lambda$ is the rate for virtual events. The algorithm samples from this process, resulting in two virtual events (dashed). In box (c) all events become potential events. The state of the root test is a queue of recent events. The state of the other test is Boolean (whether A is more recent). On the bottom is the lattice of joint states over time. Solid arrows indicate $b_i = 1$ (the event is kept). Dash arrows indicate $b_i = 0$ (the event is dropped). Each arrow's weight is as per Eq. 10. The probability of a node is the sum over all paths to the node of the product of the weights on the path (calculated by dynamic programming). In box (d) a single

TABLE I: Tests and their corresponding state representations.

Test	Example	State Representation	Property
TimeTest	Is the time between 6am and 9am?	Null	independent of b
LastEventTest	Is the last event of type A?	Boolean	Markovian
EventCountTest	Are there $\geq n$ events of type A in $[t - lag1, t - lag2]$?	A queue maintaining all the times of A between $[t - lag2, t]$, and the most recent n events between $[t - lag1, t - lag2]$.	Non-Markovian
LastStateTest	Is the last sublabel of var $A = 0$?	Boolean	Markovian
StateTest	Is the current sublabel of var $A = 0$?	Null	independent of b

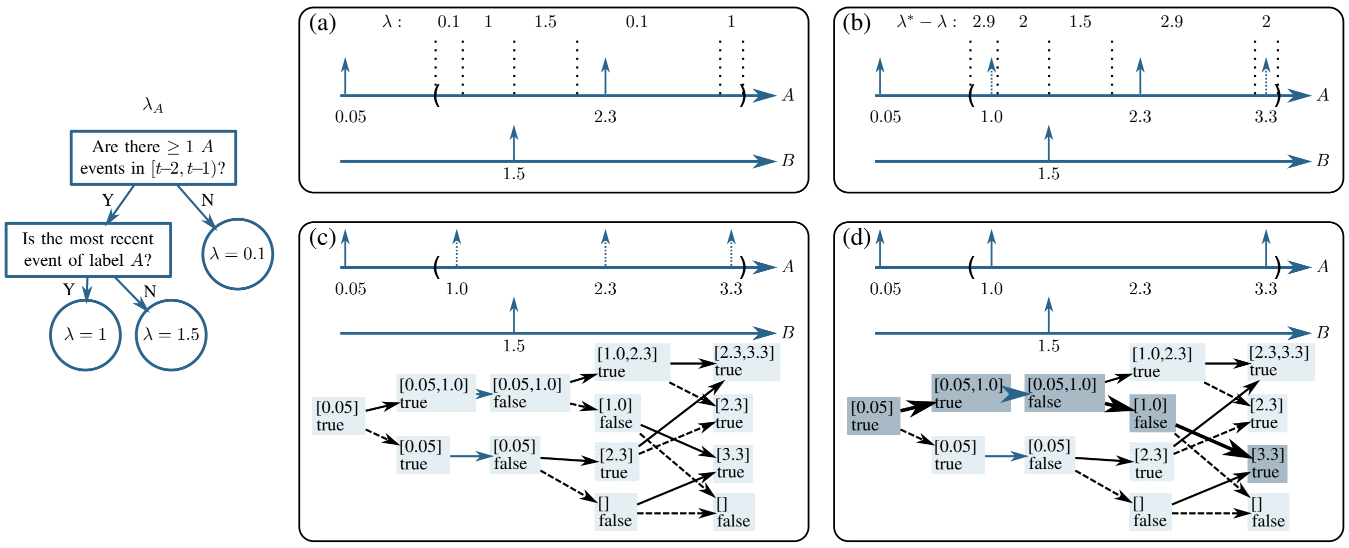


Fig. 10: Extended Example, see Section IV-G

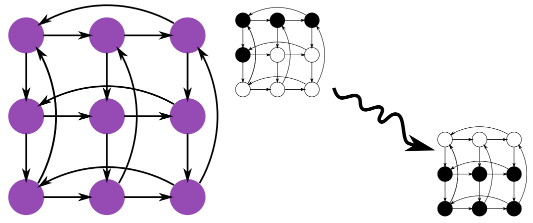


Fig. 11: The toroid network and observed patterns [12].

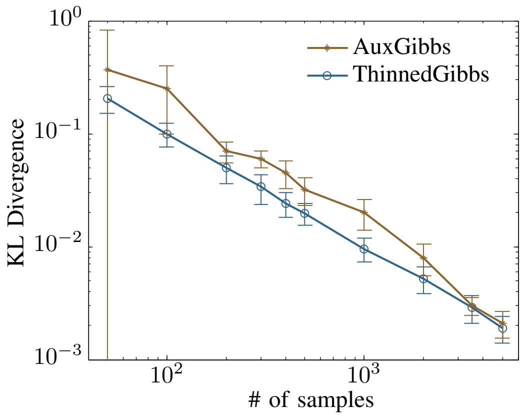


Fig. 12: Number of samples versus KL divergence for the toroid network. Both axes are on a log scale.

path is sampled with backward sampling, shown in bold. This path corresponds to keeping the first and last virtual events and dropping the middle one.

V. Experiments

A. ThinnedGibbs Validation

We perform inference with our method on both Markovian and non-Markovian models, and compare the result with the ground-truth statistics. For both we show our result converges to the correct result. Ours is the first that can successfully perform inference tasks for non-Markovian PCIMs.

1) Verification on the Ising Model: We first evaluate our method, ThinnedGibbs, on a network with Ising model dynamics. The Ising model is a well-known interaction model with applications in many fields including statistical mechanics, genetics, and neuroscience [8]. The model is Markovian and has been tested by several prior inference methods for CTBNs.

Using this model, we generate a directed toroid network structure with cycles following [12]. Nodes can take values -1 and 1, and follow their parents' states according to a coupling strength parameter (β). A rate parameter (τ) determines how fast nodes toggle between states. We test with $\beta = 0.5$ and $\tau = 2$. The network and the evidence patterns are shown in Fig. 11. The network starts from a deterministic state: at $t = 0$ variables $1 - 5$ are $+1$ and $6 - 9$ are -1. At $t = 1$, variable $1 - 3$ have switched to -1, $4 - 5$ remain $+1$, and $6 - 9$ have switched to $+1$. The nodes are not observed between $t = 0$ and $t = 1$. We query the marginal distribution of nodes at $t = 0.5$ and measure the sum of the KL-divergences of all marginals against the ground truth. We compare with the state-of-the-art CTBN Auxiliary Gibbs method [38]. Other existing methods

Algorithm 1: Resampling event l

```
input: The previous trajectory $(x_l, Y_l)$
output: The newly sampled $x_l'$
for each unobserved interval for $l$ do
    Find piecewise constant $\lambda^*(t|h)$ using $Y_l$
    Find piecewise constant $\lambda(t|h)$ using $x_l$, $Y_l$
    Sample virtual events $v_l$ with rate $\lambda^*(t|h) - \lambda(t|h)$
Let $z_l = x_l \cup v_l$, $m = |z_l|$, and $s_0$ be the initial state
AddtoProbMap($S_0$,$s_0$,1.0)
for $i \leftarrow 1$ to $m$ do
    for each $\{(s_{i-1}, \cdot) \to p\}$ in $S_{i-1}$ do
        $p_{keep} = p(E_{i-1:i}, b_i = 1 \mid s_{i-1}, E_{1:i-1})$
        $p_{drop} = p(E_{i-1:i}, b_i = 0 \mid s_{i-1}, E_{1:i-1})$
        $s_i^{keep} \leftarrow$ UpdateState($s_{i-1}$, true, $z_{l,i}$)
        $s_i^{drop} \leftarrow$ UpdateState($s_{i-1}$, false, $z_{l,i}$)
        AddtoProbMap($S_i$,($s_i^{keep}$, $z_{l,i}$), $p \times p_{keep}$)
        AddtoProbMap($S_i$,($s_i^{drop}$, $\emptyset$), $p \times p_{drop}$)
        AddtoProbMap($T_i(s_i^{keep})$, $(s_{i-1}, z_{l,i})$, $p \times p_{keep}$)
        AddtoProbMap($T_i(s_i^{drop})$, $(s_{i-1}, \emptyset)$, $p \times p_{drop}$)
Update $S_m$ by propagating until ending time
$x_l' \leftarrow \emptyset$ and $(s_m', t) \leftarrow$ SampProbMap($S_m$)
if $t \neq \emptyset$ then
    $x_l' \leftarrow x_l' \cup \{t\}$
for $i \leftarrow m-1$ to 1 do
    $(s_i', t) \leftarrow$ SampProbMap($T_{i+1}(s_{i+1}')$)
    if $t \neq \emptyset$ then
        $x_l' \leftarrow x_l' \cup \{t\}$
return $x_l'$
```

either produce similar or worse results [3]. For example, the mean field variational approach [8] produce error that is above the error range of the methods we use. We vary the sample size between 50 and 5000, and set the burn-in period to be 10% of this value. We run the experiments 100 times, and plot the means and standard deviations.

Results in Fig. 12 verify that our inference method indeed produces results that converge to the true distribution. Our method reduces to that of [38] in this Markovian model. Differences between the two lines are due to slightly different initializations of the Gibbs Markov chain and not significant.

2) Verification on a Non-Markovian Model: We further verify our method on a much more challenging non-Markovian PCIM (Fig. 13). This model contains several non-Markovian EventCountTests. We have observations for event A at $t = 0.4, 0.6, 1.8, 4.7$ and for event B at $t = 0.1, 0.2, 3.4, 3.6, 3.7$. Event A is not observed on $[2.0, 4.0)$ and event B is not observed on $[1.0, 3.0)$.

In produce ground truth, we discretized time and converted the system to a Markovian system. Note that because the time since the last A event is part of the state, as the discretization becomes finer, the state space increases. For this small example, this approach is just barely feasible. We continued to refine the discretization until the answer stabilized. The ground-truth expected total number of A events between $[0, 5]$ is 22.3206 and the expected total number of B events is 11.6161. That is, there are about 18.32 A events and 6.62 B events in the unobserved areas. Note that if the evidence is changed to have no events these numbers drop to 1.6089 and 8.6866 respectively and if the evidence after the unobserved intervals is ignored the expectations are 22.7183 and 8.6344 respectively. Therefore the evidence (both before and after the unobserved intervals) is important to incorporate in inference.

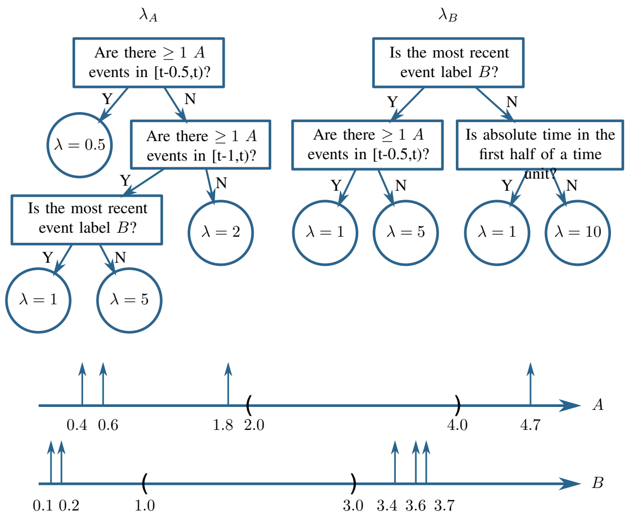

λ_A
Are there ≥ 1 A events in [t-0.5,t)?
Y
N
$\lambda = 0.5$
Are there ≥ 1 A events in [t-1,t)?
Y
N
Is the most recent event label B?
Y
N
$\lambda = 1$
$\lambda = 5$
$\lambda = 2$
λ_B
Is the most recent event label B?
Y
N
Are there ≥ 1 A events in [t-0.5,t)?
Y
N
$\lambda = 1$
$\lambda = 5$
Is absolute time in the first half of a time unit?
Y
N
$\lambda = 1$
$\lambda = 10$
0.4 0.6 1.8 2.0 4.0 4.7
A
0.1 0.2 1.0 3.0 3.4 3.6 3.7
B

Fig. 13: Non-Markovian PCIM and evidence. End time is 5.

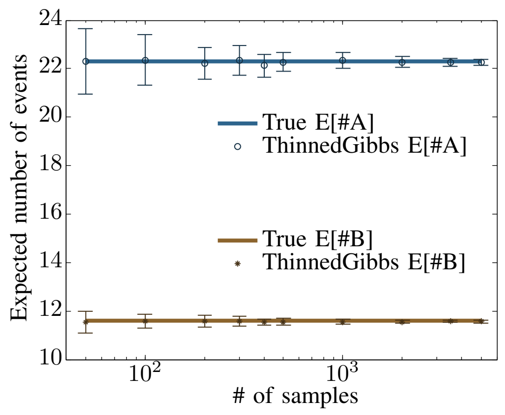


Fig. 14: Number of samples versus the inferred expected number of events. The horizontal axis is on a log scale.

We compare our inference method to the exact values, again varying the sample size between 50 and 5000 and setting the burn-in period to be 10% of this value. We ran the experiments 100 times and report the mean and standard deviation of the two expectations. Our sampler has very small bias and therefore the average values match the true value almost exactly. The variance decreases as expected, demonstrating the consistent nature of our method. See Fig. 14. We are not aware of existing methods that can perform inference on this type of model to which we could compare.

B. Experiments on Event Detection

We tested our proposed event detection approach on three challenging datasets: Hollywood [25], UCLA office [39], and PKU-MMD [28]. The Hollywood dataset is a human action dataset from movies, the UCLA dataset is from a video camera

TABLE II: Comparison of the event detection result on the Hollywood dataset. In each cell, the first number is precision and the second number is recall. Seg means the method requires video pre-segmentation.

Events	SVM(seg)	Hoai[20](seg)	SSM[7]	Ours
AnswerPhone	0.62/0.30	0.64/0.32	0.64/0.43	0.62/0.50
HugPerson	0.46/0.29	0.46/0.29	0.51/0.33	0.45/0.35
Kiss	0.39/0.46	0.40/0.48	0.44/0.59	0.45/0.55
SitDown	0.36/0.35	0.36/0.36	0.39/0.38	0.41/0.40
Overall	0.45/0.34	0.47/0.36	0.50/0.42	0.48/0.45

recording students' daily activities in an office, and the PKU-MMD dataset is a recent large-scale benchmark dataset for continuous multi-modal human action understanding.

1) Evaluation on Hollywood Dataset: The Hollywood dataset focuses on realistic human actions including AnswerPhone, Kiss, SitDown, HugPerson, StandUp, HandShake, SitUp, and GetOutCar. This dataset has two disjoint subsets with 219 video sample in the training set and 211 in the test set. We follow [7] for the experimental setting: we only focus on the first four classes to be recognized. Since the dataset contains only pre-segmented clips, new video clips of longer durations are created with enforced temporal relationships. 1-order dependency (such as SitDown-AnswerPhone) and 2-order dependency (such as HugPerson-Kiss-SitDown) are inserted. 40 such video samples were formed (see more details in [7]). To generate the visual sequences, we use mean pooling of STIP features to generate a feature vector for each 20-frame video segment, then use k-means (k is fixed to 200 for this dataset) to assign a label to each of the segment. For PCIM structure learning, we fix the bank of possible PCIM tests as EventCountTests (see Tbl. I) with $(n, lag1, lag2) \in \{1,2,3\} \times \{2,3,4,5,6\} \times \{0,1,2\}$ (omitting tests for which $lag1 \leq lag2$) and LastEventTest for all high-level and low-level event labels. We show some meaningful learned dependencies in Tbl. IV. For each run, we set the burn-in period to be 100 samples, and report the average performance of the next 20 samples.

We compare with methods that require pre-segmentation (linear SVM and [20]), as well as state-of-art method that does simultaneous segmentation and labeling ([7]). Note that results reported with pre-segmentation is biased since video segmentation is very challenging itself (for this dataset, we directly use the segments before concatenation). We perform 5-fold cross validation and the results averaged across 5 runs are reported in Tbl. II.

Our approach shows competitive performance when comparing with state-of-art methods. Though SVM and Hoai[20] are already reported on segmented videos, they still have worse performance due to the local visual ambiguities in this challenging dataset. Methods considering temporal context (SSM[7] and ours) are able to produce better results. Our approach is able to learn from the enforced temporal constraints and generate more reliable event candidates, leading to a better overall recall. Also, SSM[7] relies on training classifiers and doing inference at multiple scales and is more computationally expensive. During inference, for each video, our method generates 20 samples in less than a second,

TABLE III: High-level Event Types in the UCLA Office Datasets.

ID	Event Type	ID	Event Type
1	Enter Room	2	Exit Room
3	Sit Down	4	Stand up
5	Work on Laptop	6	Work on Paper
7	Throw Trash	8	Pour Drink
9	Pick Phone	10	Place Phone Down

TABLE IV: Examples of meaningful structures learned by PCIM on the Hollywood and UCLA dataset. Time unit used is 20 frames.

Structure Learned	Semantics
if e_3 in $[t-1, t)$ rate of $s_1 = 0.67$	The ending of "SitDown" stimulates "AnswerPhone". (Hollywood)
if w_{57} is last event and if w_{12} in $[t-1, t)$ rate of $s_2 = 0.33$	Visual words in specific order stimulate "Kiss". (Hollywood)
if s_5 in $[t-2, t)$ rate of $w_3 = 0.68$	The starting of "work on laptop" tends to generate w_3. (UCLA)
if s_3 in $[t-5, t-2)$ rate of $e_3 = 0.7$	This encodes the duration distribution of "sit down". (UCLA)
if e_5 in $[t-3, t-1)$ rate of $s_4 = 0.22$	The ending of "work on laptop" stimulates "stand up". (UCLA)

while the running time of our SSM implementation based on LibSVM[4] is in the order of minutes.

2) Evaluation on UCLA Office Dataset: The UCLA office dataset consists of 3 videos of a total length of 32 minutes, in which actors perform 10 kinds of actions in an office setting. See Tbl. III for the high-level events types and their ID numbers. (We also use mean pooling of STIP features [24] generated for each segment, but other methods are applicable.)

We perform 3-fold cross validation, using 2 videos for training and 1 video for testing. We use 20 frames as the segment length and 30 for the dictionary size (K). For PCIM structural learning and sample size, we use the same setting as in the Hollywood dataset. PCIM is able to learn 3 major kinds of meaningful dependencies that are useful for the event localization and labeling task. We show some examples in Tbl. IV. It is the combination of these rules that encodes complex dependencies in video.

We compare with state-of-art method that require pre-segmentation [50]. This method also considers temporal dependencies in a Conditional Random Field framework. Due to the complexity of inference, this method can only encode dependencies up to a fixed order. Pre-segmentation of dataset is highly nontrivial and we notice it requires manual intervention. Thus we used the segmented data from the authors of [50]. We also compare with an SVM-based approach that classifies the video segments into one of the ten high-level events plus the null event. After classification, consecutive segments of the same labels are merged to one event. This method also relaxes the necessity of video pre-segmentation, but only considers local visual evidences.

We report the results in Tbl. V. Comparing with the Hollywood dataset, this dataset contains real-world continuous videos with natural high-order temporal dependencies. SVM-based approaches can produce reasonable overall result for this dataset. For this dataset, certain dimensions in the feature vector are salient. For example, the extracted features contain location information that is salient for this dataset because the camera is static and certain events only happen at certain locations (for example, "pour drink" always happen close to the drinking machine.) SVM is able to get good result for most event classes, and tends to confuse between certain pairs of event types (i.e. "enter room" and "exit room", "stand up" and "sit down", "work on laptop" and "work on paper".) So it is a strong baseline for the event detection task on this dataset. [50] considers temporal dependencies and generate better results. We observe that, since [50] considers fixed order event dependencies, when there are null events happening between actual events (which is common in real-world videos), the learned dependencies may not be useful. Also, for datasets with high order dependencies, this model can be limited. On the other hand, PCIM explicitly models temporal dependencies of arbitrary order and is able to skip null events.

We can see that, by taking advantage of the complex temporal contexts, PCIM can produce the best precision by correcting wrong labels. Note that the discretization of image features and using uniform weights of different features (in K-Means clustering based dictionary learning) tend to lose some salient visual information. On dataset with larger intra-class variance and without dominating salient visual features, we expect PCIM to perform even better as visual evidences alone are less useful.

For recall, PCIM tends to omit event types with few training instances. PCIM can be treated as a data-driven approach and needs sufficient data to learn meaningful structures for each event types. For this particular dataset, events such as "pick up phone" are very scarce in training data, so PCIM is not able to learn meaningful structures for such event types. Then in testing, these events tend to be missing. However, since these are scarce events (also in testing data), missing them does not significantly affect the overall performance. When there is more data (such as videos from streaming surveillance that could be hundreds of hours long), we expect PCIM to mitigate such drawbacks as more instances are observed.

We show one example in Fig. 15 in which global temporal context among high-level events help to produce better result. In this example, SVM based approach tends to confuse the events "sit down" and "stand up" by only looking at local appearance information, because both events involve the actor performing actions close to the chair. PCIM, on the other hand, is able to get the correct result, by learning the temporal context that, a person should not sit down again after he had already sat down and been working on the laptop. Other typical cases that PCIM performs better involves differentiating between events such as "enter room" and "exit room."

3) Evaluation on PKU-MMD Dataset: The PKU-MMD dataset is a recent large-scale dataset for human action understanding in long continuous video sequences. It contains over 1000 long video sequences in 51 action categories, performed by 66 subjects in three camera views. In this work we focus on cross-subject evaluation. For fair comparison and evaluation, the same dataset partition setting as that in [28] was used, where the dataset is split into training and testing groups which consists of 57 and 9 subjects respectively (944 and 132 video samples respectively).

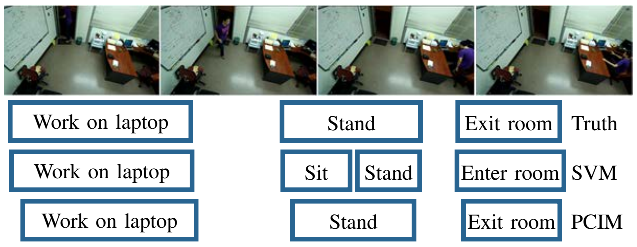


Fig. 15: Temporal context information helps to recover wrong detection by local discriminative methods. The slight time shift for PCIM is due to its continuous-time nature.

We use 20 frames as the segment length and 200 for the dictionary size (K). We compare with methods using state-of-the-art representations and temporal detection methods. Deep RGB (DR) [43] represents videos using features derived from the Temporal Segment Network (TSN). Raw skeleton (RS) is also tested in the multi-modality benchmark, but note our method only uses the videos as input. For temporal detection, we use a three stacked bidirectional LSTM (BLSTM) [49].

We report the same metrics as in [28]: $\frac{|I \cap I^*|}{|I \cup I^*|} > \theta$, where $I \cap I^*$ denotes the intersection of the predicted and ground truth intervals and $I \cup I^*$ denotes their union. Mean Average Precision (mAP) of different actions at different θ is evaluated. We directly use the evaluation tool provided in the dataset.

We report the results in Tbl. VI. Comparing with state-of-art methods that rely on more complex representations and temporal detection method, our model can produce competitive results by using RGB video input alone. Our method can produce better results at all θ levels comparing to the deep model using only RGB video input, it can also outperform the multi-modality approach (DR + RS) at higher θ levels, showing that our approach can more accurately localize events in the temporal domain. It would be interesting to extend the inference in point process for video event detection idea to multi-modality input.

VI. Conclusion

We review PCIM and how it models nonlinear dependencies in general event streams. We propose the first effective inference algorithm, ThinnedGibbs, for PCIM. Our auxiliary Gibbs sampling method effectively transforms a continuous-time problem into a discrete one. Our state-vector representation of diverging trajectories takes advantage of state merges and reduces complexity from exponential to linear for most cases. Then we show how PCIM can be used to model temporal context for event detection in video, and how event detection can be modeled as an high-level event inference problem given low-level observations. The formulation provides a generic way to model temporal context in event streams and relaxes the assumption of video pre-segmentation. It is also flexible in terms of feature extraction and dictionary learning methods.

We then validate ThinnedGibbs on several tasks, including sampling from complicated distributions with known statistics and its effectiveness in video event detection. We show our method generalizes the state-of-art inference method for

TABLE V: Comparison of the event detection result on the UCLA office dataset. Seg means the method requires video pre-segmentation. In each cell, the first number is precision and the second number is recall. See event types in Tbl. III.

Event ID	1	2	3	4	5	6	7	8	9	10	Overall
SVM	0.70/0.72	0.65/0.68	0.72/0.79	0.82/0.75	0.84/0.82	0.70/0.75	0.85/0.85	0.83/0.80	0.75/0.75	0.74/0.80	0.75/0.76
CRF[50] (seg)	0.90/0.85	0.90/0.82	0.76/0.76	0.74/0.78	0.84/0.81	0.66/0.70	0.65/0.80	0.72/0.75	0.82/0.81	0.84/0.86	0.79/0.79
Ours	0.95/0.87	0.92/0.84	0.83/0.78	0.79/0.80	0.83/0.80	0.70/0.68	0.65/0.76	0.70/0.72	0.74/0.61	0.75/0.58	0.81/0.77

TABLE VI: Mean Average Precision (mAP) comparison of the event detection result on the PKU-MMD dataset.

Method	$\theta = 0.1$	$\theta = 0.3$	$\theta = 0.5$	$\theta = 0.7$
DR + BLSTM	0.617	0.439	0.221	0.051
DR + RS + BLSTM	0.675	0.498	0.255	0.050
Ours	0.650	0.510	0.294	0.065

CTBN models. We also validate our inference idea on non-Markovian PCIMs, which is the first to do so. Then we show that the modeling of temporal context with PCIM can improve event detection performance in real-world continuous video.

REFERENCES

[1] M. R. Amer and S. Todorovic. Sum-product networks for modeling activities with stochastic structure. In *CVPR*, 2012.
[2] S. Bhattacharya, M. M. Kalayeh, R. Sukthankar, and M. Shah. Recognition of complex events: Exploiting temporal dynamic between underlying concepts. In *CVPR*, 2014.
[3] E. B. Celikkaya and C. R. Shelton. Deterministic anytime inference for stochastic continuous-time Markov processes. In *ICML*, 2014.
[4] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[5] R. Chaudhry, A. Ravichandran, G. Hager, and R. Vidal. Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In *CVPR*, 2009.
[6] Q. Chen, Y. Cai, L. Brown, A. Datta, Q. Fan, R. Feris, S. Yan, and S. Pankanti. Spatio-temporal Fisher vector coding for surveillance event detection. In *ACM Multimedia*, 2013.
[7] Y. Cheng, Q. Fan, S. Pankanti, and A. Choudhary. Temporal sequence modeling for video event detection. In *CVPR*, 2014.
[8] I. Cohn, T. El-Hay, R. Kupferman, and N. Friedman. Mean field variational approximation for continuous-time Bayesian networks. In *UAI*, 2009.
[9] C. Cotsaces, N. Nikolaidis, and I.Pitas. Video shot detection and condensed representation: a review. In *IEEE Signal Processing Magazine*, 2006.
[10] T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *AAAI*, 1988.
[11] N. Du, L. Song, M. Gomez-Rodriguez, and H. Zha. Scalable influence estimation in continuous-time diffusion networks. In *NIPS*, 2013.
[12] T. El-Hay, I. Cohn, N. Friedman, and R. Kupferman. Continuous-time belief propagation. In *ICML*, 2010.
[13] Y. Fan and C. R. Shelton. Learning continuous-time social network dynamics. In *UAI*, 2009.
[14] Y. Fan, J. Xu, and C. R. Shelton. Importance sampling for continuous time Bayesian networks. *Journal of Machine Learning Research*, 11(Aug):2115–2140, 2010.
[15] A. Golightly and D. J. Wilkinson. Bayesian parameter inference for stochastic biochemical network models using particle Markov chain Monte Carlo. *Interface Focus*, 2011.
[16] W. Grassmann. Transient solutions in Markovian queueing systems. *Computers & Operations Research*, 4(1):47–53, 1977.
[17] A. Gunawardana, C. Meek, and P. Xu. A model for temporal dependencies in event streams. In *NIPS*, 2011.
[18] M. Hasan and A. K. Roy-Chowdhury. Continuous learning of human activity models using deep nets. In *ECCV*, 2014.
[19] A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.
[20] M. Hoai, Z.-Z. Lan, and F. D. la Torre. Joint segmentation and classification of human actions in video. In *CVPR*, 2011.
[21] C. Hong, J. Yu, J. Wan, D. Tao, and M. Wang. Multimodal deep autoencoder for human pose recovery. In *IEEE Transactions on Image Processing*, 2015.
[22] Y. Kim, J. Chen, M.-C. Chang, X. Wang, E. M. Provost, and S. Lyu. Modeling transition patterns between events for temporal human action segmentations and classification. In *FG*, 2015.
[23] T. Lan, Y. Zhu, A. R. Zamir, and S. Savarese. Action recognition by hierarchical mid-level action elements. In *ICCV*, 2015.
[24] I. Laptev. On space-time interest points. *IJCV*, 64(2-3):107–123, 2005.
[25] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008.
[26] P. Lewis and G. Shedler. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979.
[27] L. Li, H. Deng, A. Dong, Y. Chang, and H. Zha. Identifying and labeling search tasks via query-based hawkes processes. In *KDD*, 2014.
[28] C. Liu, Y. Hu, Y. Li, S. Song, and J. Liu. PKU-MMD: A large scale benchmark for continuous multi-modal human action understanding. *arXiv preprint arXiv:1703.07475*, 2017.
[29] U. Nodelman, C. R. Shelton, and D. Koller. Continuous time Bayesian networks. In *UAI*, 2002.
[30] P. Over, J. Fiscus, G. Sanders, D. Joy, M. Michel, G. Awad, A. Smeaton, W. Kraaij, and G. Quénot. Trecvid 2014–an overview of the goals, tasks, data, evaluation mechanisms and metrics. In *Proceedings of TRECVID*, 2014.
[31] A. Parikh, A. Gunawardana, and C. Meek. Cojoint modeling of temporal dependencies in event streams. In *UAI Workshops*, 2012.
[32] X. Peng, C. Zou, Y. Qiao, and Q. Peng. Action recognition with stacked Fisher vectors. In *ECCV*, 2014.
[33] O. P. Popoola and K. Wang. Video-based abnormal human behavior recognition : A review. In *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2012.
[34] R. Poppe. A survey on vision-based human action recognition. In *Image and Vision Computing*, 2010.
[35] Z. Qin and C. R. Shelton. Auxiliary Gibbs sampling for inference in piecewise-constant conditional intensity models. In *UAI*, 2015.
[36] S. Rajaram, T. Graeoel, and R. Herbrich. Poisson-networks: A model for structured point process. In *AIStats*, 2005.
[37] V. Rao and Y. W. Teh. Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In *UAI*, 2011.
[38] V. Rao and Y. W. Teh. Fast MCMC sampling for Markov jump processes and extensions. *Journal of Machine Learning Research*, 14:3207–3232, 2013. arXiv:1208.4818.
[39] Z. Si, M. Pei, B. Yao, and S.-C. Zhu. Unsupervised learning of event and-or grammar and semantics from video. In *ICCV*, 2011.
[40] K. Tang, L. Fei-Fei, and D. Koller. Learning latent temporal structure for complex event detection. In *CVPR*, 2012.
[41] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *CVPR*, 2011.
[42] L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *CVPR*, 2015.
[43] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Val Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*, 2016.
[44] J. Weiss and D. Page. Forest-based point processes for event prediction from electronic health records. In *ECML-PKDD*, 2013.
[45] L. Xu, J. A. Duan, and A. Whinston. Path to purchase: A mutually exciting point process model for online advertising and conversion. In *Management Science*, 2014.
[46] S. Yeung, O. Russakovsky, G. Mori, and L. Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *CVPR*, 2016.
[47] J. Yu, X. Yang, F. Gao, and D. Tao. Deep multimodal distance metric learning using click constraints for image ranking. In *IEEE Transactions on Cybernetics*, 2017.
[48] Y. Zhang, X. Liu, M.-C. Chang, W. Ge, and T. Chen. Spatio-temporal phrases for action recognition. In *ECCV*, 2012.
[49] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen, and X. Xie. Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks. In *AAAI*, 2016.
[50] Y. Zhu, N. M. Nayak, and A. K. Roy-Chowdhury. Context-aware activity modeling using hierarchical conditional random fields. In *PAMI*, 2014.

Appendix H

Hawkes Process Inference with Missing Data

To appear as

Christian R. Shelton, Zhen Qin, and Chandini Shetty. Hawkes Process Inference with Missing Data. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Hawkes Process Inference with Missing Data

Christian R. Shelton
University of California, Riverside
cshelton@cs.ucr.edu

Zhen Qin
University of California, Riverside
zqin001@cs.ucr.edu

Chandini Shetty
University of California, Riverside
cshet001@cs.ucr.edu

Abstract

A multivariate Hawkes process is a class of marked point processes: A sample consists of a finite set of events of unbounded random size; each event has a real-valued time and a discrete-valued label (mark). It is self-excitatory: Each event causes an increase in the rate of other events (of either the same or a different label) in the (near) future. Prior work has developed methods for parameter estimation from complete samples.

However, just as unobserved variables can increase the modeling power of other probabilistic models, allowing unobserved events can increase the modeling power of point processes. In this paper we develop a method to sample over the posterior distribution of unobserved events in a multivariate Hawkes process. We demonstrate the efficacy of our approach, and its utility in improving predictive power and identifying latent structure in real-world data.

Marked Point Processes

Many applications work with records of events and their times: social networks (information propagation or network changes), computer systems (system calls, hardware failures, network events), global politics (treaties, armed conflicts), as examples. The times are real-valued, and the event types (or labels or marks) are drawn from a finite set.

The general class of marked point processes (MPPs) provide families of distributions over such data. The Poisson process is the simplest and most familiar, but the event sets from any two non-overlapping intervals of time are independent, and thus it does not provide strong modeling power. More complex MPPs include Cox processes and Poisson cluster processes.

In machine learning, a variety of structured MPPs have been developed including Poisson-networks (Rajaram, Graepel, and Herbrich 2005) and piecewise-constant conditional intensity models (PCIMs) (Gunawardana, Meek, and Xu 2011). We concentrate on Hawkes processes (Hawkes 1971). They have been used in earthquake modeling (Marsan and Lengliné 2008), finance (Linderman and Adams 2014; Bacry, Mastromatteo, and Muzy 2015), social networks (Simma and Jordan 2010; Zhou, Zha, and Song 2013; Perry and Wolfe 2013; DuBois, Butts, and Smyth 2013), influence maximization (Du et al. 2013), armed conflicts (Blundell, Heller, and Beck 2012; Mohler 2013; Linderman and Adams 2014), and topic modeling (He et al. 2015; Guo et al. 2015; Du et al. 2015).

Hidden (unobserved) variables are an essential modeling tool. They can simplify a model, be used to find hidden structure, or express the modeler's prior knowledge. Previous uses of Hawkes processes have used complete event data: all event times are observed. This paper expands the modeler's toolkit by allowing partially and fully unobserved events types. This allows hypothesizing about events associated unobserved actors, communication paths, servers, or other variables. It also allows the full use of datasets in which actors enroll or dropout at various times.

For example, we can add unobserved (but modeled) actors in a social network event process. Or, we can allow for periods of non-observation of known actors (for instance, prior to enrolling in a study). Unobserved events in other arenas might correspond to weather, news, or political events. More generally, a Hawkes process between two event-types, A and B, allows for each to self-excite or to excite the other event-type, producing clusters of events that stem from a single seed event of one type. However, if we add a third type, C, and postulate that the observed times of events for A and B result from the marginalization of a Hawkes process over all three event types, this provides a richer model of the relationship between events of types A and B, which includes relationships where A and B do not trigger each other, but both are temporally correlated (through type C).

We consider the situation in which the set of possible labels (types of events) is known. Yet, for some labels for some periods of time, it is known that any events during that time were unobserved (and during other times, all occurring events for this label were observed). This allows for completely hidden event types, as well as masking certain types during certain time intervals.

While other applications placed priors on the parameters of the Hawkes process or other parts of the model and used Gibbs sampling (or similar) over these parameters (Linderman and Adams 2014; Rasmussen 2013), no work to date has considered unobserved events to the extent that we do in this paper. Yang and Zha (2013) used mixtures of Hawkes processes and developed a variational inference method. However, all events were observed; only their assignments to mixture components were not. Xu, Luo, and Zha (2017) considered the situation in which all events prior to a specific time are unobserved (left censoring), which is a very specific case of the missingness patterns we allow. Finally, if there are no observed events of any labels and the process has an exponential kernel, a closed-form solution exists (Du et al. 2013). However, no general closed-form solution is known for the evidence patterns we consider here.

Contributions

A likelihood-weighted sampler is natural and straightforward. However, as we demonstrate in our results, it does not perform well in complex tasks. Therefore, we develop a Markov chain Monte Carlo (MCMC) method using auxiliary variables. The auxiliary variables are not only the hidden "chain" of events, which have been considered previously in Hawkes processes, but also additional "thinned" events, adapted from previous samplers for other classes of MPPs (Rao and Teh 2011; 2013; Qin and Shelton 2015). This novel combination is an efficient sampler that performs well in all tested scenarios.

We show the advantage of being able to hypothesize unseen events on real-world data. In particular, we demonstrate that, on gang homicides in Chicago, the addition of hidden labels improves prediction accuracy and reveals structure in the data that cannot be extracted without hidden events.

Multivariate Hawkes Process Background

We assume that the process begins at time $t_0 = 0$ and ends at time T. A (complete) sample from the process can be represented as $x = \{(t_1, l_1), (t_2, l_2), \ldots, (t_n, l_n)\}$ where there are n events (a random quantity) and, for notational convenience, we will let $t_{n+1} = T$ (although there is no event at T). $t_{i-1} < t_i, \forall i$ and thus t_i is the time of the ith event, and l_i is the label (mark) of the ith event. Without loss of generality, we assume that the labels are drawn from the set of integers $\{1, 2, \ldots, L\}$. We let $h_t = \{(t_i, l_i) \in x \mid t_i < t\}$ or the set of all events and labels prior to time t, analogous to the natural filtration for the process. For notational convenience, we let $\mathcal{I}_t = \{i \mid t_i < t\}$, or the set of event indices that occurred before t (also the set of event indices in h_t).

A general discrete-label MPP can be specified through the intensity function $\lambda_l(t, h_t)$ which is the rate of an event of label l at time t and is a function of the absolute time, as well as the history of events up to time t: $\lambda_l(t, h_t) = \lim_{\delta t \to 0} \Pr(\text{event of label } l \text{ in } [t, t+\delta t) \mid t, h_t)/\delta t$. The probability density of sample x is

$$p(x) = \exp\left(-\sum_l \int_0^T \lambda_l(s, h_s)\, ds\right) \prod_{i=1}^{n} \lambda_{l_i}(t_i, h_{t_i}) \ . \tag{1}$$

A (linear) multivariate Hawkes process specifies $\lambda_l(t, h_t) = \mu_l + \sum_{i \in \mathcal{I}_t} \phi_{l_i,l}(t - t_i)$ where μ_l is the base rate of events of label l and $\phi_{l,l'}(t)$, the kernel or transfer function, is the increase in the rate of label l' due to an event of label l t time units ago. We can simplify this to

$$\lambda_l(t, h_t) = \sum_{i \in \mathcal{I}_t^0} \phi_{l_i,l}(t - t_i) \tag{2}$$

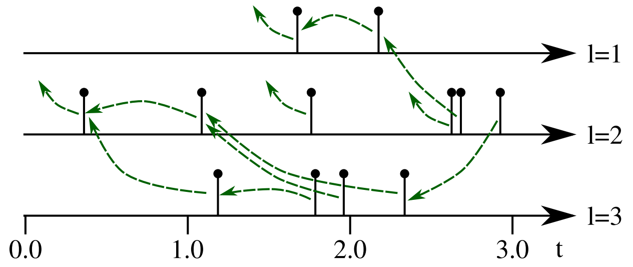


Figure 1: A sample from a multivariate Hawkes process, in black. The green dashed arrows show one possible sampling tree (the arrows point to the parents). Arrows pointing to nowhere indicate that the event was generated by the base rate (root event, $l = 0$).

if we add a special event at t_0 (a time not previously associated with an event). In particular, let $l_0 = 0$ (a new special label), let $\mathcal{I}_t^0 = \mathcal{I}_t \cup \{0\}$, and set the kernel for this new label: $\phi_{l,0}(t) = 0, \forall l$ and $\phi_{0,l}(t) = \mu_l, \forall l > 0$. This event "causes" the base rate of events for each label. For notational compactness, we will assume the kernel has been so redefined.

If we denote $\Phi_{l,l'}(t) = \int_0^t \phi_{l,l'}(s)\, ds$ and $\Phi_{l,\star}(t) = \sum_{l'} \Phi_{l,l'}(t)$, Equation 1 for a Hawkes process is

$$p(x) = \exp\left(-\sum_i \Phi_{l_i,\star}(T - t_i)\right) \prod_i \sum_{j \in \mathcal{I}_{t_i}^0} \phi_{l_j,l_i}(t_i - t_j). \tag{3}$$

Kernels

Multivariate Hawkes processes are usually defined in terms of a base kernel, $\phi(t)$, and an $L \times L$ matrix of non-negative values, M: $\phi_{l,l'}(t) = M_{l,l'}\phi(t)$. In systems with a large number of labels, M is usually sparse: Each event label only excites a small subset of other event labels. The two most common base kernels are the exponential (with parameter $\beta > 0$): $\phi(t) = e^{-\beta t}$ and the power-law (with parameters $\beta > 0$ and $\gamma > 0$): $\phi(t) = (t + \gamma)^{-(1+\beta)}$. The process is well behaved (with probability 1 there are a finite number of events on any finite interval of time) if $\lambda \int_0^\infty \phi(s)\, ds < 1$, where λ is the largest eigenvalue of M (Bacry, Mastromatteo, and Muzy 2015).

If the base kernel is an exponential, the resulting process can be viewed as a Markovian process over a continuous vector-valued state space (essentially tracking the sum of $\phi_{l,l'}(t)$ over all previous events), see Oakes (1975) and Proposition 2 of Bacry, Mastromatteo, and Muzy (2015) for more details. This can reduce the running time of sampling or likelihood sampling (but not our MCMC sampler). We used this improved method for exponential kernels in our experimental results, but otherwise the details are not relevant and are left for the supplementary material.

Unconditional Sampling

The Poisson superposition principle states that the union of events from two independent Poisson processes is itself a Poisson process whose rate function is the sum of the rate functions of the two underlying processes. Equation 2 shows the rate as the sum of independent rates (one for each past event). Each event generates a set of *children* events independently (at time t the rate of an event is the sum of the rates of any previous event generating a child at time t). This view of a Hawkes process is well established (Hawkes and Oakes 1974) and critical to the development of our sampler.

The sampling algorithm can therefore be recursive in nature. We start with the special label 0 at time 0 and proceed until time T, sampling "children" events from the base rates. Each of these children recursively generates its own events from the kernel rate function. An event (t, l) generates a set of events independently for each other label l' from the inhomogeneous Poisson process with rate $\lambda(t') = M_{l,l'}\phi(t' - t), \forall t' > t$.

This recursive structure forms a tree of events; each event has a parent whose kernel rate function was used to generate it. Figure 1 show one possible sample, along with the (normally discarded) information about which events recursively generated which other events in dashed green.

Posterior Sampling

Our goal is to reason about the distribution of such a process, conditioned on observations of the events for only some labels over some intervals of time. That is, we assume that we observe all elements of x that fall within certain observed label-time ranges.

We let the observational evidence, $z = (z^{(x)}, z^{(o)})$, be such a partial sample which specifies the events during certain time intervals for certain labels. $z^{(x)} = \{(t_1, l_1), \ldots, (t_m, l_m)\}$ is the set of observed events ($z^{(x)} \subseteq x$), analogous to x, but specifying $m \leq n$ events. $z^{(o)} = \{(r_1^s, r_1^e, l_1), \ldots, (r_k^s, r_k^e, l_k)\}$ specifies k observed intervals. In particular, the ith element of $z^{(o)}$ specifies that all events of label l_i on $t \in [r_i^s, r_i^e)$ were observed. We assume the data are missing at random: $z^{(o)} \perp x$. We let $z_l^{(o)}$ denote the union of the intervals over which label l is observed.

Our goal is to estimate $p(x \mid z)$. Note that while the unconditional sampler could sample each set of children independently, conditioned on evidence, these samples are no longer independent. The conditional process $p(x \mid z)$ is not a Hawkes process.

Markov Chain Monte Carlo

A likelihood-weighted sampler is straight-forward (just restrict the sample generation to agree with the evidence), but (as shown later), this method does not perform well on problems of even moderate size.

Auxiliary Variables We use Metropolis-Hastings sampling (Metropolis et al. 1953; Hastings 1970) on the tree representation of the unconditional sampler from above. For an effective MCMC sampler, we introduce two sets of auxiliary variables.

The first auxiliary variable set records the parent structure of the recursive, generational view of a Hawkes process (the green arrows in Figure 1). This set has been used in prior work (Veen and Schoenberg 2008; Marsan and Lengliné 2008) to estimate the parameters of a Hawkes process with *complete data* using expectation-maximization.

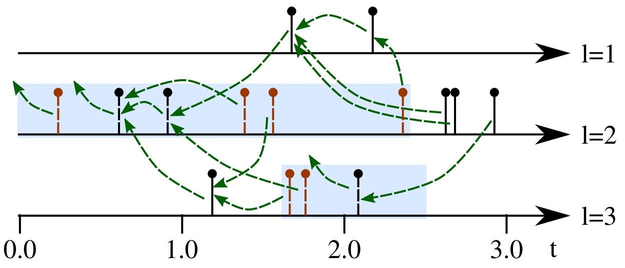


Figure 2: An example sample from the full MCMC process given missing data: Label 2 is unobserved until time 2.4 and label 3 is unobserved from time 1.6 until time 2.5. Black events are x (evidence are solid, sampled are dashed). The parent auxiliary variables, a and $\tilde{a}$, are in green. The virtual events, $\tilde{x}$, are in orange.

The second auxiliary variable set consists of "virtual events," potential (but not realized) times for events, similar to those used for continuous-time Markov processes (Rao and Teh 2011; 2013) and PCIMs (Qin and Shelton 2015). The virtual events are fast to generate (as they are not part of the evidence) and provide a finite number of potential events to turn into a real event (through a sampler move). They are resampled (through other sampler moves) to allow for potentially any real-valued event time. In this way, the times for posterior events that might couple two observed events can be search efficiently without considering the uncountably infinite number of potential event times.

While each type has been used before, they have not previously been used together. The parent auxiliary variables allow for a decoupling of the likelihood (much like mixture-assignment variables in clustering). And, the virtual events are needed to change the dimensionality of the sampling space in a computationally simple fashion. Together, they allow us to tackle Hawkes processes with unobserved events, which no previous sampling method has addressed.

Parent Auxiliary Variables Let $a = \{a_1, \ldots, a_n\}$ where a_i is the index of the parent of event with index i. If we keep track of this information, the unconditional sampler (from above) can be seen as generating samples from $p(x, a)$, whose marginal $p(x)$ is the desired prior distribution (that is, if we throw away the green arrows in Figure 1 we have a sample over labels and times). For notation, let $c_{i,l}$ denote the children of event i that have label l: $\{(t_j, l) \in x \mid a_j = i\}$ and $c_i = \bigcup_l c_{i,l}$. The joint distribution over x and a has only multiplicative terms:

$$p(x, a) = \prod_i \phi_{l_{a_i}, l_i}(t_i - t_{a_i}) \exp\left(-\Phi_{l_i,\star}(T - t_i)\right) .$$

It is straight-forward to show that $\sum_a p(x, a) = p(x)$ (see Equation 3).

Virtual Event Auxiliary Variables These virtual events are potential (but not realized) children for each of the real events (and the special "root event"). They do not generate their own children (real or virtual) and are not part of x. Let $\tilde{x} = \{(\tilde{t}_1, \tilde{l}_1), \ldots, (\tilde{t}_{\tilde{n}}, \tilde{l}_{\tilde{n}})\}$ denote the set of virtual events. Let $\tilde{a}_i$ denote the index of the parent (a real event) of the ith virtual event. Let $\tilde{c}_{i,l}$ and $\tilde{c}_i$, be analogous to $c_{i,l}$ and c_i, but for the ith virtual event.

The unconditional Hawkes process generates events in $c_{i,l}$ from a Poisson process with rate $\phi_{l_i,l}(t-t_i)$. Our distribution including the auxiliary variables generates the events in $\tilde{c}_{i,l}$ from a Poisson process with rate $\kappa \cdot \phi_{l_i,l}(t-t_i)$. Therefore, unconditioned on evidence, by the Poisson superposition principle, the events in $c_{i,l} \cup \tilde{c}_{i,l}$ are drawn from a Poisson process with rate $(\kappa+1) \cdot \phi_{l_i,l}(t-t_i)$. Furthermore, given an event is from this union, it is a virtual event with probability $\kappa/(\kappa+1)$, regardless of its time.

Complete Joint Process Our total auxiliary process is over $x, a, \tilde{x}$, and $\tilde{a}$. Its marginal over x is the same as the original multivariate Hawkes process. Its joint is

$$p(x, a, \tilde{x}, \tilde{a}) = \prod_{i=1}^{n} \exp\left[-(\kappa+1)\Phi_{l_i,\star}(T-t_i)\right] \times \prod_{i=1}^{n} \phi_{l_{a_i},l_i}(t_i - t_{a_i}) \prod_{i=1}^{\tilde{n}} \kappa \cdot \phi_{l_{\tilde{a}_i},\tilde{l}_i}(\tilde{t}_i - t_{\tilde{a}_i}) \ . \tag{4}$$

MCMC Sampler Figure 2 shows a sample from this process. We build an MCMC sampler over the dashed items in this figure: events in x that are in the unobserved periods, a and $\tilde{a}$ (the parent sets of all events), and $\tilde{x}$. We constrain $\tilde{x}$ to also only contain events in during the unobserved periods to reduce computation time because virtual events during an observed period can never become real.

Our sampler maintains a state of $x, a, \tilde{x}$, and $\tilde{a}$. There are four types of events: the root event (label 0), evidence events, sampled events, and virtual events. The first three are members of x and the last of $\tilde{x}$. The root and evidence events cannot be changed, but all of the other variables can. At each iteration of the sampler, we pick an event uniformly at random from $x \cup \tilde{x}$. We then select one of the following three "moves" uniformly at random. If the move does not apply to the event, we consider it a self-transition in the Markov chain.

Move 1: Virtual Children This only applies to events from x. Let the event have index i. For this move, we consider replacing the current set $\tilde{c}_i$ with a new set, $\tilde{c}'_i$ drawn from a Poisson process with rate $\kappa \cdot \phi_{l_i,l}(t-t_i)$. This changes the dimension of the distribution (by adding new or removing variables in $\tilde{x}$ and $\tilde{a}$). However, these new variables are sampled without regard to the current state of the process, and therefore the "Jacobian" correction from reversible-jump MCMC is 1 for this case (Green 1995). This is the only type of dimension-changing move (also used as part of the other moves).

Move 2: Virtualness Assume the event is the pair (t, l). If the event is virtual, we propose changing it to be real (moving it from $\tilde{x}$ to x) and sampling a set of virtual children as in Move 1. The likelihood ratio corresponds to moving a term from the second product to the first product in Equation 4, times the probability of sampling these new virtual children. This second part cancels with the proposal likelihood (same as above). Again, we have a correction ratio corresponding to the change in the number of events.

If the event is not virtual, to assure reversibility, we only propose a change if it is neither the root nor an evidence event, and if it has no non-virtual children. In this case, we propose moving it from x to $\tilde{x}$ and removing its virtual children from $\tilde{x}$.

Move 3: Parent This move only applies to non-root events from x. Resampling parents from among x is straight-forward but misses the gains possible by allowing evidence (or sampled events) to "reach back" and suggest possible events earlier in the timeline. Therefore, we allow virtual events to be selected as parents. If one is, it becomes real, and we sample virtual children for this new parent.

While local parent changes are appealing (they can be proposed in constant time), they are difficult to make reversible because they might insert virtual events between the old and new parent, thus rendering the reverse move non-local. Therefore, we sample a new proposed parent from any event earlier in time. Let the event whose parent is to be resampled be (t, l); let $\tilde{h}_t$ be all previous virtual events; and let $H_t = h_t \cup \tilde{h}_t$. We propose $(t'_p, l'_p) \in H_t$ as the new parent with probability proportional to $w(t'_p, l'_p) = \phi_{l,l'_p}(t - t'_p)$. If $(t'_p, l'_p) \in \tilde{x}$, then our proposal includes moving it to x and sampling (as in Move 1) new virtual children for it, $\tilde{c}'$. If the proposed change would leave the old parent with no real children and it is not the root event, then we propose to change the old parent to be virtual (and remove its children) with probability $\kappa/(\kappa+1)$.

The moves were designed, for the most part, to simplify the acceptance ratios. The exact form of the ratios are straightforward, but tedious to derive. We leave them to the supplemental material.

Synthetic Data Experiments

For evaluation, we used an exponential and a power-law kernel. We generated two different sizes of problems, each in an "easy" and "hard" version. We then tested the time-accuracy trade-offs of the base likelihood weighting and our MCMC method on each of these eight combinations.

Because the algorithms are samplers, estimating the expectation of any function of the sample is possible. We chose the number of events for a particular set of labels because this query is simple and directly related to the average *posterior* rate, which is important in kernel parameter estimation.

We chose the exponential kernel with $\beta = 1$ and the power-law kernel with $\beta = 1$ (inverse squared decay) and $\gamma = 1$, so they integrate to the same quantity. The power-law kernel has a heavier tail, however, and so more of its power will

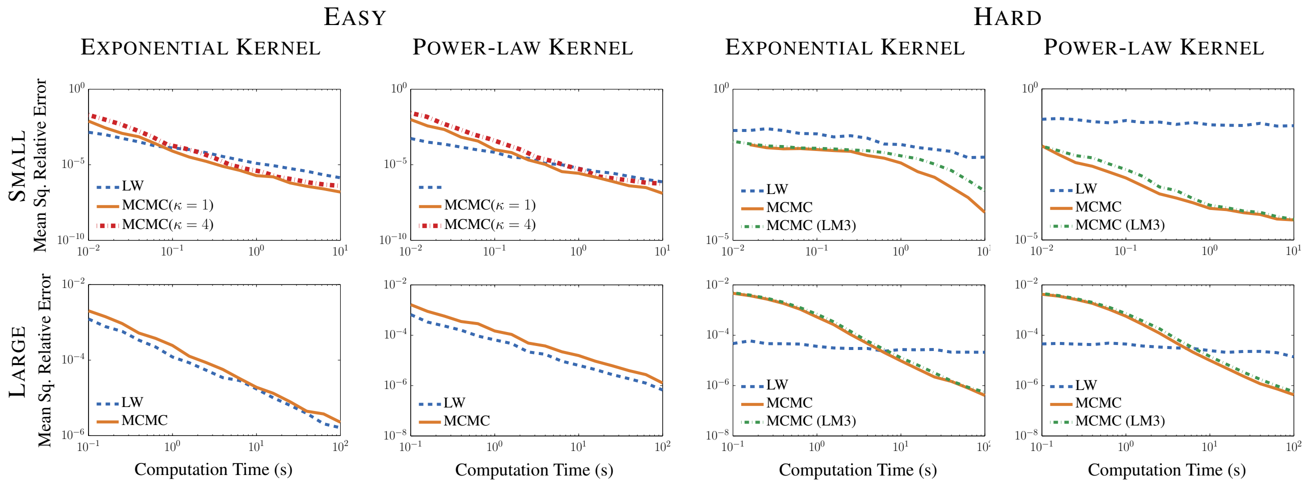


Figure 3: Results on synthetic problems. Both likelihood weighting (LW) and our MCMC sampler perform well on easy problems. LW fails on hard problems, while our MCMC sampler continues to performs well. LM3 is a limited version of "move 3."

fall after time T. We describe the small and large synthetic problems in more detail in the supplementary material. The small problems consist of chains of labels (each affecting the next one or two). The large problems consist of labels connected via a graph with a power-law degree distribution. All problems have evidence on approximately half of the labels.

Methods

Each combination of kernel, problem size, and problem difficulty fixed a particular input process, evidence, and query. We chose 16 geometrically evenly spaced computation times and independently ran the algorithms 200 independent times for each of these computation times, stopping the sampling when the algorithm had reached the computational time limit. For the MCMC method, this computational budget included the burn-in time, which we set to be 1000 iterations for the easy problems and 5000 iterations for hard problems.

Running for a fixed computational time, instead of a fixed number of iterations, can bias the resulting estimator toward samples with smaller numbers of events (because they take less time, generally, to generate). However, our experience did not show this was a factor in these experiments.

For each problem and computational budget, we used the 200 estimates to compute the mean squared relative error (the relative variance). We computed the true values by running the MCMC sampler for 5 hours. Both sampling algorithms can be easily parallelized, but, so as not to complicate the comparison, we ran all experiments on a single core. The single tunable parameter is κ. We set it to 1, following the methodology of similar samplers (Rao and Teh 2011; Qin and Shelton 2015). We illustrate the effect of changing κ on the small-easy problem. We also tried removing the ability of the MCMC "move 3" to select a previously virtual parent and illustrate its effect on the hard problems (limited move 3).

Problem			MCMC	LW	
diff	size	kernel	samp/sec	samp/sec	eff/sec
E	S	exp	3.32×10^6	4.67×10^5	2.02×10^3
E	S	pow	2.84×10^6	5.71×10^5	4.17×10^3
E	L	exp	8.65×10^5	8.53×10^2	1.87×10^2
E	L	pow	7.66×10^5	9.58×10^2	4.34×10^2
H	S	exp	5.61×10^5	1.23×10^4	0.30
H	S	pow	7.14×10^5	1.75×10^4	0.16
H	L	exp	3.51×10^5	1.04×10^2	0.15
H	L	pow	2.12×10^5	1.83×10^1	0.22

Table 1: Generation speed statistics for both algorithms. Column samp/sec lists the average number of samples taken per second (a single step for MCMC and a full sample for LW). Column eff/sec lists the effective number of samples for LW (the average effective sample size across all runs with the longest runtime).

Results

Figure 3 shows the accuracy-time trade-off for the algorithms. On the easy problems (Figure 3, left), both algorithms perform well. The likelihood weighting method has fewer samples and many fewer effective samples, measured as $(\sum_i w_i)^2 / \sum_i w_i^2$, see Table 1. However, the performance suggests this is compensated by the independence of each sample, unlike MCMC. For the small instance, we also show the very small effect of changing κ to 4. We found similarly small differences for adjustments of κ between 0.5 and 8 for all of the experimental designs tested.

On the hard problems (Figure 3, right), things are different. Our MCMC method performs well with expected $O(1/n)$ reduction in variance. The exponential kernel for the small problem requires more than the provided burn-in time (a full burn-in is not always possible in 0.01 seconds, hence the missing point). Thus, the $O(1/n)$ behavior does not start until after 1 second. If we remove the ability of the MCMC sampler to select a previously virtual event as a parent (lim-

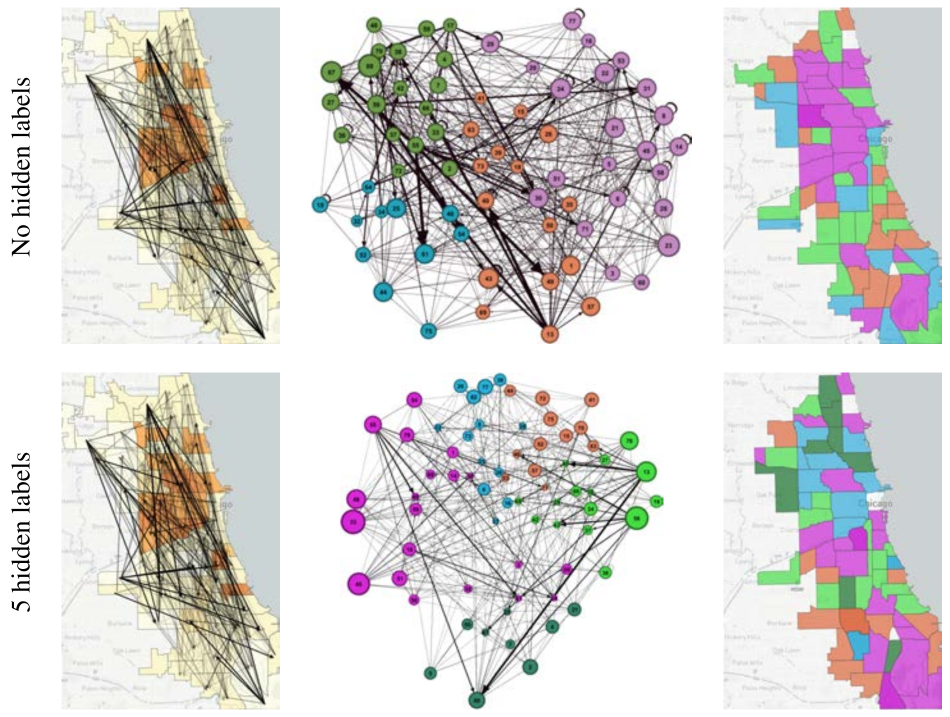


Figure 4: Results for crime data, without hidden event types (top) and with hidden event types (bottom). Background region color on left map shows the (area-normalized) background rates (μ) of the estimated model. Estimated model network (M) shown in three ways: (left) as super-imposed on map with arc darkness proportional to weight in M, (middle) as a filtered graph, laid out to demonstrate automatically found clusters, and (right) the same clusters shown geographically.

ited move 3), the performance degrades when there are more events and the base rate is small, and therefore it does not naturally generate real events. This can be seen in the small hard problem with an exponential kernel. Note that adding this ability never hurts the performance.

The likelihood weighting performs terribly on the hard problems. We do not even see the expected $O(1/n)$ performance. This is because very few samples with any significant weight are generated as seen in Table 1 in the "eff/sec" column. While the bias and variance of this estimator do decrease as $O(1/n)$, 200 runs is not sufficient to demonstrate this effect. Almost never, in all of the 200 runs and all of the samples taken in each of these runs, does the sampler generate a sample that is even remotely likely under the posterior distribution.

All sampling code exploits the sparsity of M.

Crime Data Experiments

To demonstrate the utility of unobserved variables, we used homicide data provided by the Chicago Police Department from 1965 through 1995 (Block, Block, and Illinois Criminal Justice Information Authority 2005) filtered to only those events reported to be gang-related. This follows the same methodology as Papachristos (2009). We treat each gang-related homicide as an event with a label corresponding to the community area in which it occurred. There are 77 different Chicago communities identified in the dataset and 2195 events. We did not supply any proximity information about the regions to the model or algorithm.

While a self-excitatory process would seem to be a good fit to gang-related homicides, previous uses of a plain Hawkes process failed to find structure in the data. Others have used a Hawkes process as part of a more complex model to find structure. For instance, Linderman and Adams (2014) placed hierarchical priors over the parameters to produce a clustering of communities. We show that using a plain Hawkes process with unobserved event types (labels) will also find structure, and does better at prediction, relative to a plain Hawkes process.

Methods

We first used a Hawkes process with an exponential kernel to model the 77 fully observed labels (one for each community of Chicago). Because there are no missing data, our sampling method reduces to sampling over the latent generational structure (a). We use this as the expectation step in a Monte Carlo expectation maximization procedure (Wei and Tanner 1990) to estimate the parameters: M (matrix of inter-label weights), β (decay rate of kernel), and μ (vector of base rate for each label). We included an L_1 regularizer on the elements of M with strength set via a search over powers of 10. We let $\kappa = 1$ in our sampler. For the maximization step, given a collection of samples and β, M and μ can be solved in closed form (see supplementary material). We use a 1-dimensional line search to find β. This method is essentially the same as previous EM methods for Hawkes processes with fully observed labels

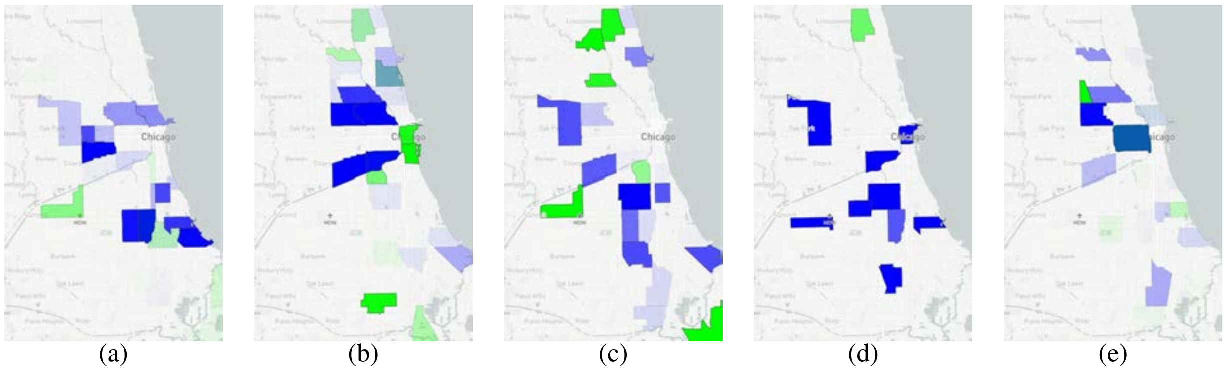


Figure 5: The strength of connection to (in green) geographic labels and from (in blue) geographic labels each of the five hidden event labels.

but hidden branching structure (Veen and Schoenberg 2008; Marsan and Lengliné 2008).

To demonstrate the advantage of hidden events, we repeated the same process, but with five extra process labels, each completely unobserved. These can be viewed as missing variables (event sequences) with unspecified semantics. They can be used by the (unaided) learning procedure to simplify the structure, similar to how hidden variables in a Bayesian network can simplify the distribution's representation. Note each hidden event type is not a single scalar random variable, but rather a full (unobserved) sequence of event times.

These processes are allowed arbitrary connections with each other and the 77 observed labels. To encourage their use in the model, we clamped the elements of M corresponding to connections between two observed labels to 0 for the first half of the EM iterations.

Finally, we reran the models, training only on crimes from 1993 and 1994. We then tested on the last year of data (1995), advancing time one event at a time and predicting the location of the next event (the time of the next event is almost always in the next zero to three days not not interesting to estimate). There are 215 events in 1995 on 171 different days. We credited a model if the region with the highest probability of having the next event corresponded to any of the regions with an event on that day. We used our MCMC method to do the forecasting.

Results

The prediction accuracy for the model with five hidden event types was 22% higher (6.5% versus 5.3%). However, more interesting was the hidden event model's ability to capture sociological structure in the data.

The EM algorithm was very stable, producing very similar results on multiple runs, despite different starting values and random seeds. The resulting β value is $\frac{1}{28.6\,days}$ for the model with no hidden event types and $\frac{1}{48.2\,days}$ for the model with hidden event types. The M and μ values are shown in Figure 4. The result is messy (despite tuning the regularization) and thus consistent with previous attempts. The network among the observed variables is slightly sparser for the model with hidden event types, but it is hard to tell from this figure. We clustered the regions (see supplemental material for method) to try to find meaningful clusters, shown in Figure 4. While some are geographically reasonable, they do not reflect the major connectivity of the model.

Yet, the hidden event connections of the model with 5 hidden labels do directly reveal known gang-related structure. Figure 5 shows the connections in M to and from the five hidden labels. Each plot, therefore, shows regions (in blue) that tend to get "triggered" together (from the same hidden event) and regions (in green) that tend to trigger these hidden events. These results correlate well with previous studies. Block and Block (1993) mapped out Chicago gang crime in the late 1980s. Neither the base rates of models nor clustering of the M matrix (Figure 4) identify the gang areas in south Chicago. However, two of our hidden label connections, Figure 5 (c,d), identify the primary areas in south Chicago also noted in Exhibit 1 of Block and Block (1993). The cluster in Figure 5(b) matches the first cluster (that of highest crime rate) identified by Linderman and Adams (2014). Further, the "Street Gang Turfs" of 1991 highlighted in Exhibit 4 of Block and Block (1993) are also identified in Figure 5(e).

Summary

The code for general inference in Hawkes processes with optional parallelization, as well as the wrapper code to run the exact experiments done here and gather the results will be supplied in a public GitHub repository.

We developed a reversible-jump MCMC sampler for Hawkes processes. It allows the use of hidden event types in Hawkes processes, providing flexibility to modelers. We demonstrated the utility of such hidden events by using them to more accurately predict locations, and to find meaningful clusters of regions in Chicago crime data.

Acknowledgements This work was supported by the Defense Advanced Research Projects Agency (FA8750-14-2-0010) and the National Science Foundation (IIS 1510741).

References

Bacry, E.; Mastromatteo, I.; and Muzy, J.-F. 2015. Hawkes processes in finance. Market Microstructure and Liquidity 1(1).

Block, C. R., and Block, R. 1993. Street gang crime in chicago. Research in brief, National Institute of Justice.

Block, C. R.; Block, R. L.; and Illinois Criminal Justice Information Authority. 2005. Homicides in Chicago, 1965–1995. Technical Report ICPSR 6399, Inter-university Consortium for Political and Social Research.

Blundell, C.; Heller, K. A.; and Beck, J. M. 2012. Modelling reciprocating relationships with Hawkes processes. In NIPS.

Du, N.; Song, L.; Gomez-Rodriguez, M.; and Zha, H. 2013. Scalable influence estimation in continuous-time diffusion networks. In NIPS.

Du, N.; Farajtabar, M.; Ahmed, A.; Smola, A. J.; and Song, L. 2015. Dirichlet-Hawkes processes with applications to clustering continuous-time document streams. In KDD.

DuBois, C.; Butts, C. T.; and Smyth, P. 2013. Stochastic blockmodeling of relational event dynamics. In AI-STATS.

Green, P. J. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. Biometrika 82(4):711–732.

Gunawardana, A.; Meek, C.; and Xu, P. 2011. A model for temporal dependencies in event streams. In NIPS.

Guo, F.; Blundell, C.; Wallach, H.; and Heller, K. 2015. The Bayesian echo chamber: Modeling social influence via linguistic accommodation. In AI-STATS.

Hastings, W. K. 1970. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57:97–109.

Hawkes, A. G., and Oakes, D. 1974. A cluster process representation of a self-exciting process. Journal of Applied Probability 11(3):493–503.

Hawkes, A. G. 1971. Spectra of some self-exciting and mutually exciting point processes. Biometrika 58(1):83–90.

He, X.; Rekatsinas, T.; Foulds, J.; Getoor, L.; and Liu, Y. 2015. HawkesTopic: A joint model for network inference and topic modeling from text-based cascades. In ICML.

Linderman, S. W., and Adams, R. P. 2014. Discovering latent network structure in point process data. In ICML.

Marsan, D., and Lengliné, O. 2008. Extending earthquakes' reach through cascading. Science 319:1076–1079.

Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; and Teller, E. 1953. Equations of state calculations by fast computing machines. Journal of Chemical Physics 21(6):1087–1092.

Mohler, G. 2013. Modeling and estimation of multi-source clustering in crime and security data. The Annals of Applied Statistics 7(3):1525–1539.

Oakes, D. 1975. The Markovian self-exciting process. Journal of Applied Probability 12(1):69–77.

Papachristos, A. V. 2009. Murder by structure: Dominance relations and the social structure of gang homicide. American Journal of Sociology 115(1):74–128.

Perry, P. O., and Wolfe, P. J. 2013. Point process modelling for directed interaction networks. Journal of the Royal Statistical Society: Series B 75(5):821–849.

Qin, Z., and Shelton, C. R. 2015. Auxiliary Gibbs sampling for inference in piecewise-constant conditional intensity models. In UAI.

Rajaram, S.; Graepel, T.; and Herbrich, R. 2005. Poisson networks: A model for structured point processes. In Proceedings of the AI STATS 2005 Workshop.

Rao, V., and Teh, Y. W. 2011. Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In UAI.

Rao, V., and Teh, Y. W. 2013. Fast MCMC sampling for Markov jump processes and extensions. Journal of Machine Learning Research 14:3207–3232.

Rasmussen, J. G. 2013. Bayesian inference for Hawkes processes. Methodology and Computing in Applied Probability 15(3):623–642.

Simma, A., and Jordan, M. 2010. Modeling events with cascades of Poisson processes. In UAI.

Veen, A., and Schoenberg, F. P. 2008. Estimation of space-time branching process models in seismology using an EM-type algorithm. Journal of the American Statistical Association 103(482):614–624.

Wei, G. C. G., and Tanner, M. A. 1990. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. Journal of the American Statistical Association 85(411):699–704.

Xu, H.; Luo, D.; and Zha, H. 2017. Learning Hawkes processes from short doubly-censored event sequences. In ICML.

Yang, S.-H., and Zha, H. 2013. Mixture of mutually exciting processes for viral diffusion. In ICML.

Zhou, K.; Zha, H.; and Song, L. 2013. Learning triggering kernels for multi-dimensional Hawkes processes. In ICML, volume 28.

List of Symbols, Abbreviations, and Acronyms

Term	Explanation
AUROC	Area under receiver operating characteristic
AuxGibbs	Auxiliary Gibbs sampler
BDD	Binary decision diagram
BST	Binary search tree
BTF	Brightness transfer function
CIM	Conditional intensity model
CHLA	Children's Hospital Los Angeles
CT-NOR	Continuous-time noisy or
CTBN	Continuous-time Bayesian network
CTBN-RLE	Continuous-time Bayesian network reasoning and learning engine
CTMP	Continuous-time Markov chain
CPU	Central processing unit
CRF	Conditional random field
DAT	Data association-based tracking
DBN	Dynamic Bayesian network
EHR	Electronic health record
EM	Expectation-Maximization
EV*MDD	Edge-valued matrix decision diagram
FFBS	Forward filtering, backward sampling algorithm
GPU	Graphics processing unit
HCI	Human-computer interaction
HMM	Hidden Markov model
HP	Hawkes process
HSV	Hue-saturation-value
ICU	Intensive care unit
LSTM	Long short-term memory
MAP	Maximum a posteriori
MCMC	Markov chain Monte Carlo
MIL	Multiple instance learning
MIMIC	Medical information mart for intensive care

Term	Explanation
MJP	Markov jump process
MPP	Marked point process
MTBDD	Multi-terminal binary decision diagram
MTGP	Multi-task Gaussian process
PCIM	Piecewise-constant conditional intensity model
PDB	Protein data bank
PIM	Pediatric index of mortality (score)
PRISM	Pediatric risk of mortality (score)
RNN	Recurrent neural network
ROC	Receiver operating characteristic
SAPS	Simplified acute physiology score
SVM	Support vector machine
TOP	Time-ordered products
TTOP	Tree of time-ordered products